

**AMERICAN NATIONAL STANDARD**

**ANSI/ISA-100.11a-2011**

**Wireless systems for industrial automation:  
Process control and related applications**

**Approved 4 May 2011**

ANSI/ISA-100.11a-2011

Wireless systems for industrial automation: Process control and related applications

ISBN: 978-1-936007-96-7

Copyright © 2011 by the International Society of Automation (ISA). All rights reserved. Not for resale. Printed in the United States of America. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means (electronic mechanical, photocopying, recording, or otherwise), without the prior written permission of the Publisher.

ISA  
67 Alexander Drive  
P.O. Box 12277  
Research Triangle Park, North Carolina 27709

## Preface

This preface, as well as all footnotes and annexes, is included for information purposes and is not part of ANSI/ISA-100.11a-2011.

This document has been prepared as part of the service of ISA toward a goal of uniformity in the field of instrumentation. To be of real value, this document should not be static but should be subject to periodic review. Toward this end, the Society welcomes all comments and criticisms and asks that they be addressed to the Secretary, Standards and Practices Board; ISA; 67 Alexander Drive; P. O. Box 12277; Research Triangle Park, NC 27709; Telephone (919) 549-8411; Fax (919) 549-8288; E-mail: [standards@isa.org](mailto:standards@isa.org).

The ISA Standards and Practices Department is aware of the growing need for attention to the metric system of units in general, and the International System of Units (SI) in particular, in the preparation of instrumentation standards. The Department is further aware of the benefits to USA users of ISA standards of incorporating suitable references to the SI (and the metric system) in their business and professional dealings with other countries. Toward this end, this Department will endeavor to introduce SI-acceptable metric units in all new and revised standards, recommended practices, and technical reports to the greatest extent possible. *Standard for Use of the International System of Units (SI): The Modern Metric System*, published by the American Society for Testing & Materials as IEEE/ASTM SI 10-97, and future revisions, will be the reference guide for definitions, symbols, abbreviations, and conversion factors.

It is the policy of ISA to encourage and welcome the participation of all concerned individuals and interests in the development of ISA standards, recommended practices, and technical reports. Participation in the ISA standards-making process by an individual in no way constitutes endorsement by the employer of that individual, of ISA, or of any of the standards, recommended practices, and technical reports that ISA develops.

**CAUTION — ISA DOES NOT TAKE ANY POSITION WITH RESPECT TO THE EXISTENCE OR VALIDITY OF ANY PATENT RIGHTS ASSERTED IN CONNECTION WITH THIS DOCUMENT, AND ISA DISCLAIMS LIABILITY FOR THE INFRINGEMENT OF ANY PATENT RESULTING FROM THE USE OF THIS DOCUMENT. USERS ARE ADVISED THAT DETERMINATION OF THE VALIDITY OF ANY PATENT RIGHTS, AND THE RISK OF INFRINGEMENT OF SUCH RIGHTS, IS ENTIRELY THEIR OWN RESPONSIBILITY.**

**PURSUANT TO ISA'S PATENT POLICY, ONE OR MORE PATENT HOLDERS OR PATENT APPLICANTS MAY HAVE DISCLOSED PATENTS THAT COULD BE INFRINGED BY USE OF THIS DOCUMENT AND EXECUTED A LETTER OF ASSURANCE COMMITTING TO THE GRANTING OF A LICENSE ON A WORLDWIDE, NON-DISCRIMINATORY BASIS, WITH A FAIR AND REASONABLE ROYALTY RATE AND FAIR AND REASONABLE TERMS AND CONDITIONS. FOR MORE INFORMATION ON SUCH DISCLOSURES AND LETTERS OF ASSURANCE, CONTACT ISA OR VISIT [WWW.ISA.ORG/STANDARDSPATENTS](http://WWW.ISA.ORG/STANDARDSPATENTS).**

**OTHER PATENTS OR PATENT CLAIMS MAY EXIST FOR WHICH A DISCLOSURE OR LETTER OF ASSURANCE HAS NOT BEEN RECEIVED. ISA IS NOT RESPONSIBLE FOR IDENTIFYING PATENTS OR PATENT APPLICATIONS FOR WHICH A LICENSE MAY BE REQUIRED, FOR CONDUCTING INQUIRIES INTO THE LEGAL VALIDITY OR SCOPE OF PATENTS, OR DETERMINING WHETHER ANY LICENSING TERMS OR CONDITIONS PROVIDED IN CONNECTION WITH SUBMISSION OF A LETTER OF ASSURANCE, IF ANY, OR IN ANY LICENSING AGREEMENTS ARE REASONABLE OR NON-DISCRIMINATORY.**

**ISA REQUESTS THAT ANYONE REVIEWING THIS DOCUMENT WHO IS AWARE OF ANY PATENTS THAT MAY IMPACT IMPLEMENTATION OF THE DOCUMENT NOTIFY THE ISA STANDARDS AND PRACTICES DEPARTMENT OF THE PATENT AND ITS OWNER.**

**ADDITIONALLY, THE USE OF THIS DOCUMENT MAY INVOLVE HAZARDOUS MATERIALS, OPERATIONS OR EQUIPMENT. THE DOCUMENT CANNOT ANTICIPATE ALL POSSIBLE APPLICATIONS OR ADDRESS ALL POSSIBLE SAFETY ISSUES**

**ASSOCIATED WITH USE IN HAZARDOUS CONDITIONS. THE USER OF THIS DOCUMENT MUST EXERCISE SOUND PROFESSIONAL JUDGMENT CONCERNING ITS USE AND APPLICABILITY UNDER THE USER'S PARTICULAR CIRCUMSTANCES. THE USER MUST ALSO CONSIDER THE APPLICABILITY OF ANY GOVERNMENTAL REGULATORY LIMITATIONS AND ESTABLISHED SAFETY AND HEALTH PRACTICES BEFORE IMPLEMENTING THIS DOCUMENT.**

**THE USER OF THIS DOCUMENT SHOULD BE AWARE THAT THIS DOCUMENT MAY BE IMPACTED BY ELECTRONIC SECURITY ISSUES. THE COMMITTEE HAS NOT YET ADDRESSED THE POTENTIAL ISSUES IN THIS VERSION.**

INTRODUCTION .....	28
REVISION HISTORY .....	30
1 Scope .....	31
2 Normative references .....	31
3 Terms, definitions, abbreviated terms, acronyms, and conventions .....	32
3.1 (N)-layer and other terms and definitions from the open systems interconnection basic reference model .....	32
3.2 Other terms and definitions .....	37
3.3 Symbols .....	52
3.4 Abbreviated terms and acronyms .....	53
3.5 IEC service table conventions .....	59
4 Overview .....	61
4.1 General .....	61
4.2 Interoperability .....	61
4.3 Quality of service .....	61
4.4 Worldwide applicability .....	61
4.5 Network architecture .....	61
4.6 Network characteristics .....	63
5 Systems .....	68
5.1 General .....	68
5.2 Devices .....	68
5.3 Networks .....	74
5.4 Protocol suite structure .....	83
5.5 Data flow .....	84
5.6 Time reference .....	89
5.7 Firmware upgrades .....	90
5.8 Wireless backbones and other infrastructures .....	90
6 System management .....	91
6.1 General .....	91
6.2 Device management application process .....	93
6.3 System manager .....	114
7 Security .....	161
7.1 General .....	161
7.2 Security services .....	161
7.3 Frame security .....	165
7.4 The join process .....	194
7.5 Session establishment .....	222
7.6 Key update .....	229
7.7 Security manager functionality .....	236
7.8 Security policies .....	238
7.9 Security functions available to the application layer .....	240
7.10 Security statistics collection, threat detection, and reporting .....	243
7.11 Device security management object functionality .....	243
8 Physical layer .....	251
8.1 General .....	251
8.2 Default physical layer .....	251

9	Data link layer .....	253
9.1	General .....	253
9.2	Data link layer data service access point .....	315
9.3	Data frames and acknowledgements .....	317
9.4	Data link layer management information base.....	339
9.5	Data link layer methods .....	381
9.6	Data link layer alerts .....	383
10	Network layer .....	386
10.1	General .....	386
10.2	Network layer functionality overview .....	386
10.3	Network layer data services .....	404
10.4	Network layer management object .....	406
10.5	Network layer protocol data unit formats .....	413
11	Transport layer.....	421
11.1	General .....	421
11.2	Transport layer reference model .....	421
11.3	Transport security sub-layer .....	422
11.4	Transport data entity.....	422
11.5	Transport layer protocol data unit encoding .....	426
11.6	Transport layer model.....	428
12	Application layer .....	438
12.1	General .....	438
12.2	Energy considerations .....	438
12.3	Legacy control system considerations.....	438
12.4	Introduction to object-oriented modeling .....	439
12.5	Object model .....	441
12.6	Object attribute model .....	442
12.7	Method model.....	444
12.8	Alert model .....	444
12.9	Alarm state model.....	445
12.10	Event state model.....	446
12.11	Alert reporting.....	446
12.12	Communication interaction model .....	448
12.13	Application layer addressing .....	457
12.14	Management objects.....	461
12.15	User objects .....	461
12.16	Data types .....	491
12.17	Application services provided by application sub-layer.....	497
12.18	Application layer flow use to lower layer services .....	530
12.19	Application layer management .....	531
12.20	Process control industry standard data structures .....	551
12.21	Additional tables .....	554
12.22	Coding.....	555
12.23	Syntax .....	574
12.24	Detailed coding examples (INFORMATIVE) .....	587
13	Gateway.....	589
13.1	General .....	589
13.2	Service access point.....	593

13.3	Protocol .....	632
14	Provisioning .....	651
14.1	General .....	651
14.2	Terms and definitions .....	651
14.3	Provisioning procedures .....	653
14.4	Pre-installed symmetric keys .....	653
14.5	Provisioning using out-of-band mechanisms .....	654
14.6	Provisioning networks.....	654
14.7	State transition diagrams .....	656
14.8	Device management application protocol objects for provisioning .....	661
14.9	Management objects.....	663
14.10	Device provisioning service object .....	668
14.11	Provisioning functions (INFORMATIVE) .....	677
Annex A (INFORMATIVE)	Protocol implementation conformance statement proforma .....	680
A.1	Introduction .....	680
A.2	System .....	682
A.3	System management .....	685
A.4	Security manager .....	688
A.5	Physical layer .....	689
A.6	Data link layer .....	691
A.7	Network layer .....	692
A.8	Transport layer .....	694
A.9	Application layer .....	695
A.10	Gateway .....	703
A.11	Provisioning.....	705
Annex B (NORMATIVE)	Role profiles .....	707
B.1	Introduction .....	707
B.2	System .....	708
B.3	System manager.....	708
B.4	Security manager .....	708
B.5	Physical layer .....	709
B.6	Data link layer .....	710
B.7	Network layer .....	715
B.8	Transport layer .....	716
B.9	Application layer .....	716
B.10	Gateway .....	716
B.11	Provisioning.....	717
Annex C (INFORMATIVE)	Background information .....	718
C.1	Industrial needs .....	718
C.2	Usage classes .....	718
C.3	Other uploading and downloading- alarms (human or automated action) .....	719
C.4	The open systems interconnection basic reference model .....	720
Annex D (NORMATIVE)	Configuration defaults .....	723
D.1	General .....	723
D.2	System management .....	723
D.3	Security .....	724
D.4	Data link layer .....	724
D.5	Network layer .....	725

D.6	Transport layer .....	726
D.7	Application layer .....	726
D.8	Gateway .....	728
D.9	Provisioning .....	728
Annex E (INFORMATIVE)	Use of backbone networks .....	730
E.1	General .....	730
E.2	Recommended characteristics .....	730
E.3	Internet protocol backbones .....	730
Annex F (NORMATIVE)	Basic security concepts – Notation and representation .....	732
F.1	Strings and string operations .....	732
F.2	Integers, octets, and their representation .....	732
F.3	Entities .....	732
Annex G (INFORMATIVE)	Using certificate chains for over-the-air provisioning .....	733
Annex H (NORMATIVE)	Security building blocks .....	734
H.1	Symmetric key cryptographic building blocks .....	734
H.2	Asymmetric key cryptographic building blocks .....	735
H.3	Keying information .....	735
H.4	Key agreement schemes .....	736
H.5	Keying information schemes .....	737
H.6	Challenge domain parameter generation and validation .....	738
H.7	Challenge validation primitive .....	738
H.8	Secret key generation (SKG) primitive .....	739
H.9	Block-cipher-based cryptographic hash function .....	739
H.10	Elliptic curve cryptography manual certificate scheme .....	740
Annex I (INFORMATIVE)	Definition templates .....	743
I.1	Object type template .....	743
I.2	Standard object attributes template .....	743
I.3	Standard object methods .....	744
I.4	Standard object alert reporting template .....	745
I.5	Data structure definition .....	746
Annex J (INFORMATIVE)	Operations on attributes .....	747
J.1	Operations on attributes .....	747
J.2	Synchronized cutover .....	750
Annex K (NORMATIVE)	Standard object types .....	751
Annex L (INFORMATIVE)	Standard data types .....	757
Annex M (NORMATIVE)	Protocol identification values .....	759
Annex N (INFORMATIVE)	Tunneling and native object mapping .....	760
N.1	Overview .....	760
N.2	Tunneling .....	760
N.3	Foreign protocol application communication .....	760
N.4	Native object mapping .....	761
N.5	Tunneling and native object mapping tradeoffs .....	761
Annex O (INFORMATIVE)	Generic protocol translation .....	762
O.1	Overview .....	762
O.2	Publish .....	762
O.3	Subscribe .....	763
O.4	Client .....	763

O.5 Server .....	764
Annex P (INFORMATIVE) Gateway service access point adaptations for this standard .....	766
P.1 General .....	766
P.2 Parameters .....	766
P.3 Session .....	766
P.4 Lease .....	766
P.5 Device list report .....	767
P.6 Topology report .....	767
P.7 Schedule report .....	767
P.8 Device health report .....	767
P.9 Neighbor health report .....	767
P.10 Network health report .....	767
P.11 Time .....	767
P.12 Client/server .....	767
P.13 Publish/subscribe .....	768
P.14 Bulk transfer .....	769
P.15 Alert .....	769
P.16 Gateway configuration .....	769
P.17 Device configuration .....	769
Annex Q (INFORMATIVE) Gateway service access point adaptations for WirelessHART® .....	770
Q.1 General .....	770
Q.2 Parameters .....	771
Q.3 Session .....	771
Q.4 Lease .....	771
Q.5 Device list report .....	772
Q.6 Topology report .....	772
Q.7 Schedule report .....	772
Q.8 Device health report .....	773
Q.9 Neighbor health report .....	773
Q.10 Network health report .....	774
Q.11 Time .....	774
Q.12 Client/server .....	774
Q.13 Publish/subscribe .....	775
Q.14 Bulk transfer .....	776
Q.15 Alert .....	776
Q.16 Gateway configuration .....	777
Q.17 Device configuration .....	777
Annex R (INFORMATIVE) Host system interface to standard-compliant devices via a gateway .....	778
R.1 Background .....	778
R.2 Device application data integration with host systems .....	779
R.3 Host system configuration tool .....	779
R.4 Field device / distributed control systems integration .....	781
R.5 Gateway .....	781
R.6 Asset management application support .....	782
Annex S (INFORMATIVE) Symmetric Key Operation Test Vectors .....	783
S.1 DPDU samples .....	783
S.2 TPDU samples .....	784

Annex T (INFORMATIVE) Data link header and network header for join requests .....	786
T.1 Overview .....	786
T.2 MAC header (MHR) .....	786
T.3 DL header (DHR).....	786
T.4 NL header .....	786
Bibliography.....	788

Table 1 – Standard management object types in DMAP .....	95
Table 2 – Meta_Data_Attribute data structure .....	97
Table 3 – Alert types for communication diagnostic category .....	98
Table 4 – Alert types for security alert category .....	98
Table 5 – Alert types for device diagnostic alert category.....	98
Table 6 – Alert types for process alert category .....	99
Table 7 – ARMO attributes.....	100
Table 8 – ARMO alerts.....	103
Table 9 – Alarm_Recovery method .....	104
Table 10 – DMO attributes .....	107
Table 11 – DMO alerts .....	113
Table 12 – System management object types .....	116
Table 13 – DSO attributes.....	118
Table 14 – Address_Translation_Row data structure .....	118
Table 15 – Read_Address_Row method.....	120
Table 16 – Input argument usage.....	121
Table 17 – Output argument usage .....	121
Table 18 – Attributes of SMO in system manager .....	123
Table 19 – Proxy_System_Manager_Join method .....	124
Table 20 – Proxy_System_Manager_Contract method .....	125
Table 21 – Effect of Different Join Commands on Attribute Sets.....	127
Table 22 – Attributes of DMSO in system manager .....	127
Table 23 – System_Manager_Join method .....	128
Table 24 – System_Manager_Contract method .....	129
Table 25 – Attributes of STSO in system manager .....	133
Table 26 – Attributes of SCO in system manager .....	136
Table 27 – SCO method for contract establishment, modification, or renewal.....	138
Table 28 – Input argument usage.....	143
Table 29 – Output argument usage .....	144
Table 30 – Contract_Data data structure .....	147
Table 31 – New_Device_Contract_Response data structure .....	150
Table 32 – SCO method for contract termination, deactivation and reactivation .....	156
Table 33 – DMO method to terminate contract .....	156
Table 34 – DMO method to modify contract .....	158
Table 35 – Security levels .....	165

Table 36 – Structure of the security control field .....	166
Table 37 – Sec.D pduPrep.Request elements .....	170
Table 38 – Sec.D pduPrep.Response elements.....	171
Table 39 – Sec.DL AckCheck.Request elements .....	172
Table 40 – Sec. DL AckCheck.Response elements .....	172
Table 41 – Sec.D pduCheck.Request elements .....	173
Table 42 – Sec.D pduCheck.Response elements .....	174
Table 43 – Sec.DL AckPrep.Request elements .....	174
Table 44 – Sec.DL AckPrep.Response elements.....	175
Table 45 – Structure of the WISN DPDU nonce .....	176
Table 46 – Structure of the 32-bit truncated TAI time .....	176
Table 47 – TSS “pseudo-header” structure.....	181
Table 48 – Sec.T pduOutCheck.Request elements.....	184
Table 49 – Sec.T pduOutCheck.Response elements .....	184
Table 50 – Sec.T pduSecure.Request elements .....	185
Table 51 – Sec. T pduSecure.Response elements .....	186
Table 52 – Sec.T pduInCheck.Request elements .....	187
Table 53 – Sec.T pduInCheck.Response elements.....	187
Table 54 – Sec.T pduVerify.Request elements .....	188
Table 55 – Sec.T pduVerify.Response elements .....	189
Table 56 – Structure of TL security header .....	189
Table 57 – Structure of the TPDU nonce .....	190
Table 58 – Structure of 32-bit nominal TAI time .....	190
Table 59 – Proxy_Security_Sym_Join method.....	201
Table 60 – Security_Sym_Join method .....	201
Table 61 – Security_Confirm method .....	202
Table 62 – Security_Sym_Join_Request data structure .....	202
Table 63 – Security_Sym_Join_Response data structure .....	203
Table 64 – Structure of compressed Security level field .....	204
Table 65 – Master key security level .....	205
Table 66 – Structure of KeyHardLifeSpan field.....	205
Table 67 – Security_Sym_Confirm data structure .....	206
Table 68 – Implicit certificate format .....	207
Table 69 – Usage_Serial structure .....	208
Table 70 – Proxy_Security_Pub_Join method .....	213
Table 71 – Security_Pub_Join method .....	213
Table 72 – Proxy_Security_Pub_Confirm method.....	214
Table 73 – Security_Pub_Confirm method .....	214
Table 74 – Network_Information_Confirmation method .....	215
Table 75 – Format of asymmetric join request internal structure.....	216
Table 76 – Format of the protocol control field .....	216
Table 77 – Format of asymmetric join response internal structure .....	217
Table 78 – Format of first join confirmation internal structure .....	218

Table 79 – Format of join confirmation response internal structure .....	219
Table 80 – Join process and device lifetime state machine .....	221
Table 81 – Security_New_Session method .....	226
Table 82 – Security_New_Session_Request data structure .....	227
Table 83 – Security_New_Session_Response data structure .....	228
Table 84 – New_Key method .....	230
Table 85 – Security_Key_and_Policies data structure .....	232
Table 86 – Security_Key_Update_Status data structure .....	234
Table 87 – Session and DL key state transition .....	235
Table 88 – Attributes of PSMO in the system manager .....	236
Table 89 – Structure of Policy field .....	238
Table 90 – Key types .....	239
Table 91 – Key usage .....	239
Table 92 – Granularity .....	239
Table 93 – Device security management object attributes .....	244
Table 94 – KeyDescriptor (INFORMATIVE) .....	246
Table 95 – TL KeyLookupData OctetString fields .....	247
Table 96 – Delete key method .....	248
Table 97 – Key_Policy_Update method .....	249
Table 98 – DSMO Alerts .....	250
Table 99 – Timing requirements .....	251
Table 100 – Graph table on ND20 .....	257
Table 101 – Graph table on ND21 .....	257
Table 102 – Approximating nominal timing with 32 kHz clock (INFORMATIVE) .....	284
Table 103 – DL_Config_Info structure .....	308
Table 104 – DD-DATA.request parameters .....	315
Table 105 – DD-DATA.confirm parameters .....	316
Table 106 – Value set for status parameter .....	316
Table 107 – DD-DATA.indication parameters .....	317
Table 108 – ExtDLUInt, one-octet variant .....	319
Table 109 – ExtDLUInt, two-octet variant .....	319
Table 110 – Data frame MHR .....	320
Table 111 – DHDR frame control octet .....	321
Table 112 – Data frame DMXHR .....	321
Table 113 – DROUT structure, compressed variant .....	322
Table 114 – DROUT structure, uncompressed variant .....	323
Table 115 – DADDR structure .....	324
Table 116 – Acknowledgement frame MHR .....	325
Table 117 – Acknowledgement frame DHR .....	326
Table 118 – DHR ACK/NACK frame control .....	326
Table 119 – Advertisement DAUX structure .....	328
Table 120 – Advertisement selections elements .....	328
Table 121 – Advertisement selections .....	329

Table 122 – Advertisement time synchronization elements .....	329
Table 123 – Advertisement time synchronization structure .....	329
Table 124 – Join superframe information subfields.....	330
Table 125 – Join superframe information structure .....	330
Table 126 – Superframe derived from advertisement .....	331
Table 127 – Join information elements .....	331
Table 128 – Join information structure .....	332
Table 129 – Defaults for links created from advertisements .....	333
Table 130 – dlmo.Neighbor entry created from advertisements .....	334
Table 131 – dlmo.Graph entry created from advertisements.....	334
Table 132 – dlmo.Route entry created from advertisements .....	334
Table 133 – Solicitation header subfields .....	336
Table 134 – Solicitation header structure .....	336
Table 135 – Solicitation DAUX fields .....	336
Table 136 – Solicitation DAUX structure .....	336
Table 137 – Activate link DAUX fields .....	338
Table 138 – Activate link DAUX structure.....	338
Table 139 – Reporting received signal quality DAUX fields .....	338
Table 140 – Report received signal quality DAUX structure.....	338
Table 141 – DLMO attributes .....	340
Table 142 – Subnet filter octets .....	347
Table 143 – dlmo.TaiAdjust OctetString fields .....	348
Table 144 – dlmo.TaiAdjust OctetString structure .....	348
Table 145 – dlmo.EnergyDesign OctetString fields .....	348
Table 146 – dlmo.EnergyDesign OctetString structure .....	348
Table 147 – dlmo.DeviceCapability OctetString fields .....	349
Table 148 – dlmo.DeviceCapability OctetString structure .....	349
Table 149 – dlmo.DiscoveryAlert fields .....	351
Table 150 – dlmo.DiscoveryAlert structure .....	351
Table 151 – dlmo.Candidates OctetString fields .....	352
Table 152 – dlmo.Candidates structure .....	352
Table 153 – dlmo.SmoothFactors OctetString fields .....	353
Table 154 – dlmo.SmoothFactors structure .....	353
Table 155 – dlmo.QueuePriority fields.....	354
Table 156 – dlmo.QueuePriority structure .....	354
Table 157 – dlmo.ChannelDiag fields .....	355
Table 158 – dlmo.ChannelDiag structure .....	355
Table 159 – dlmo.Ch fields .....	357
Table 160 – dlmo.Ch structure .....	357
Table 161 – Receive template fields .....	360
Table 162 – Receive template structure .....	360
Table 163 – Transmit template fields .....	361
Table 164 – Transmit template structure .....	361

Table 165 – Default receive template .....	362
Table 166 – Default transmit template.....	362
Table 167 – Default receive template for scanning .....	362
Table 168 – dlmo.Neighbor fields .....	364
Table 169 – dlmo.Neighbor structure .....	364
Table 170 – ExtendGraph fields .....	366
Table 171 – ExtGraph structure .....	366
Table 172 – dlmo.NeighborDiagReset fields.....	366
Table 173 – dlmo.NeighborDiagReset structure .....	366
Table 174 – dlmo.Superframe fields .....	367
Table 175 – dlmo.Superframe structure .....	368
Table 176 –dlmo.Superframeldle fields .....	371
Table 177 – dlmo.Superframeldle structure .....	371
Table 178 – dlmo.Graph.....	372
Table 179 – dlmo.Graph structure .....	372
Table 180 – dlmo.Link fields .....	373
Table 181 – dlmo.Link structure .....	374
Table 182 – dlmo.Link[].Type structure .....	375
Table 183 – Allowed dlmo.Link[].Type combinations .....	376
Table 184 – Values for dlmo.Link[].Schedule .....	377
Table 185 – dlmo.Route fields.....	377
Table 186 – dlmo.Route structure .....	378
Table 187 – dlmo.NeighborDiag fields.....	379
Table 188 – Diagnostic Summary OctetString fields .....	379
Table 189 – Diagnostic Summary OctetString structure.....	380
Table 190 – Diagnostic ClockDetail OctetString fields .....	380
Table 191 – Diagnostic ClockDetail OctetString structure.....	381
Table 192 – Read_Row method .....	381
Table 193 – Write_Row method .....	382
Table 194 – Write_Row_Now method.....	382
Table 195 – dlmo.AlertPolicy fields .....	383
Table 196 – dlmo.AlertPolicy OctetString structure .....	383
Table 197 – DL_Connectivity alert .....	384
Table 198 – DL_Connectivity alert OctetString .....	384
Table 199 – NeighborDiscovery alert .....	385
Table 200 – Link local address structure .....	387
Table 201 – Address translation table (ATT) .....	387
Table 202 – Example of a routing table .....	393
Table 203 – N-DATA.request elements .....	404
Table 204 – N-DATA.confirm elements.....	405
Table 205 – N-DATA.indication elements .....	406
Table 206 – NLMO attributes .....	407
Table 207 – Contract table structure .....	410

Table 208 – Route table elements .....	411
Table 209 – Address translation table structure .....	411
Table 210 – NLMO structured MIB manipulation methods .....	412
Table 211 – Alert to indicate dropped PDU/PDU error .....	413
Table 212 – Common header patterns .....	414
Table 213 – Basic network layer header format.....	415
Table 214 – Contract-enabled network layer header format.....	416
Table 215 – 6LoWPAN_IPHC encoding format.....	417
Table 216 – IPv6 network layer header format .....	418
Table 217 – Full network layer header in the DL .....	419
Table 218 – Network layer header format for fragmented NPDUs .....	419
Table 219 – First fragment header format .....	419
Table 220 – Second and subsequent fragment header format .....	420
Table 221 – UDP header encoding.....	423
Table 222 – UDP LowPAN_NHC encoding .....	427
Table 223 – Optimal UDP header encoding.....	427
Table 224 – UDP header encoding with checksum .....	428
Table 225 – T-DATA.request elements .....	429
Table 226 – T-DATA.confirm elements .....	430
Table 227 – T-DATA.confirm status codes .....	430
Table 228 – T-DATA.indication elements.....	431
Table 229 – TLMO attributes.....	432
Table 230 – Transport layer management object methods – Reset.....	434
Table 231 – Transport layer management object methods – Halt .....	434
Table 232 – Transport layer management object methods – PortRangeInfo .....	435
Table 233 – Transport layer management object methods – GetPortInfo.....	435
Table 234 – Transport layer management object methods – GetNextPortInfo.....	436
Table 235 – Transport layer management object alert types – Illegal use of port.....	436
Table 236 – Transport layer management object alert types – TPDU received on unregistered port .....	437
Table 237 – Transport layer management object alert types – TPDU does not match security policies .....	437
Table 238 – State table for alarm transitions .....	445
Table 239 – State table for event transitions .....	446
Table 240 – UAP management object attributes .....	462
Table 241 – State table for UAP management object .....	463
Table 242 – UAP management object methods .....	464
Table 243 – Alert receiving object attributes .....	465
Table 244 – State table for handling an AlertReport reception.....	465
Table 245 – AlertReceiving object methods.....	466
Table 246 – UploadDownload object attributes .....	468
Table 247 – UploadDownload object methods .....	471
Table 248 – UploadDownload object StartDownload method .....	472
Table 249 – UploadDownload object DownloadData method .....	473

Table 250 – UploadDownload object EndDownload method .....	475
Table 251 – UploadDownload object StartUpload method .....	476
Table 252 – UploadDownload object UploadData method .....	477
Table 253 – UploadDownload object EndUpload method .....	478
Table 254 – Download state table for unicast operation mode .....	479
Table 255 – Upload state table for unicast operation mode .....	481
Table 256 – Concentrator object attributes.....	484
Table 257 – Concentrator object methods .....	485
Table 258 – Dispersion object attributes .....	486
Table 259 – Dispersion object methods.....	487
Table 260 – Tunnel object attributes .....	488
Table 261 – Tunnel object methods .....	490
Table 262 – Interface object attributes .....	491
Table 263 – Interface object methods .....	491
Table 264 – Data type: ObjectAttributeIndexAndSize .....	492
Table 265 – Data type: Communication association endpoint .....	493
Table 266 – Data type: Communication contract data .....	494
Table 267 – Data type: Alert communication endpoint.....	495
Table 268 – Data type: Tunnel endpoint.....	495
Table 269 – Data type: Alert report descriptor .....	496
Table 270 – Data type: Process control alarm report descriptor for analog with single reference condition .....	496
Table 271 – Data type: ObjectIDandType .....	497
Table 272 – Data type: Unscheduled correspondent .....	497
Table 273 – AL services.....	498
Table 274 – Publish service .....	501
Table 275 – Read service .....	508
Table 276 – Write service .....	512
Table 277 – Execute service .....	515
Table 278 – AlertReport service .....	521
Table 279 – AlertAcknowledge service .....	524
Table 280 – Tunnel service .....	527
Table 281 – Application flow characteristics .....	530
Table 282 – Application service primitive to transport service primitive mapping .....	531
Table 283 – ASLMO attributes .....	533
Table 284 – Application sub-layer management object methods .....	534
Table 285 –Reset method .....	535
Table 286 – ASLMO alerts .....	536
Table 287 – Analog input object attributes .....	539
Table 288 – Analog input object methods.....	540
Table 289 – Analog input alerts.....	541
Table 290 – Analog output attributes.....	543
Table 291 – Analog output object methods.....	544

Table 292 – Analog output alerts .....	545
Table 293 – Binary input object attributes .....	547
Table 294 – Binary input object methods.....	548
Table 295 – Binary input alerts .....	548
Table 296 – Binary output attributes.....	549
Table 297 – Binary output object methods .....	550
Table 298 – Binary output alerts .....	550
Table 299 – Status octet .....	551
Table 300 – Data type: Process value and status for analog value .....	552
Table 301 – Data type: Process value and status for binary value .....	552
Table 302 – Data type: Process control mode .....	553
Table 303 – Data type: Process control mode bitstring.....	553
Table 304 – Data type: Process control scaling .....	554
Table 305 – Process control standard objects .....	554
Table 306 – Services .....	554
Table 307 – Application messaging format .....	555
Table 308 – Concatenated APDUs in a single TSDU.....	555
Table 309 – Object addressing.....	556
Table 310 – Four-bit addressing mode APDU header construction .....	556
Table 311 – Eight-bit addressing mode APDU header construction .....	556
Table 312 – Sixteen-bit addressing mode APDU header construction .....	557
Table 313 – Inferred addressing use case example.....	557
Table 314 – Inferred addressing mode APDU header construction .....	557
Table 315 – Six-bit attribute identifier, not indexed.....	558
Table 316 – Six-bit attribute identifier, singly indexed, with seven-bit index.....	558
Table 317 – Six-bit attribute identifier, singly indexed, with fifteen-bit index .....	558
Table 318 – Six-bit attribute identifier, doubly indexed, with two seven-bit indices .....	559
Table 319 – Six-bit attribute identifier, doubly indexed, with two fifteen-bit indices .....	559
Table 320 – Six-bit attribute identifier, doubly indexed, with first index seven-bits long and second index fifteen bits long .....	559
Table 321 – Six-bit attribute bit attribute identifier, doubly indexed, with first index fifteen bits long and second index seven bits long .....	559
Table 322 – Twelve-bit attribute identifier, not indexed .....	559
Table 323 – Twelve-bit attribute identifier, singly indexed with seven-bit index.....	560
Table 324 – Twelve-bit attribute identifier, singly indexed with fifteen bit identifier .....	560
Table 325 – Twelve-bit attribute identifier, doubly indexed with two seven bit indices .....	560
Table 326 – Twelve-bit attribute identifier, doubly indexed with two fifteen bit indices .....	560
Table 327 – Twelve-bit attribute identifier, doubly indexed with first index seven-bits long and second index fifteen-bits long .....	560
Table 328 – Twelve-bit attribute identifier, doubly indexed with the first index fifteen bits long and the second index seven bits long .....	561
Table 329 – Twelve-bit attribute identifier, reserved form .....	561
Table 330 – Coding rules for read service request .....	561
Table 331 – Coding rules for read service response with seven bit length field .....	561

Table 332 – Coding rules for read service response with fifteen-bit length field .....	562
Table 333 – Coding rules for write service request with seven bit length field .....	562
Table 334 – Coding rules for write service request with fifteen-bit length field .....	562
Table 335 – Coding rules for write service response .....	562
Table 336 – Coding rules for execute service request with seven-bit length field.....	563
Table 337 – Coding rules for execute service request with fifteen-bit length field .....	563
Table 338 – Coding rules for execute service response with 7-bit length field .....	563
Table 339 – Coding rules for execute service response with 15-bit length field .....	563
Table 340 – Coding rules for tunnel service request with seven-bit length field .....	564
Table 341 – Coding rules for tunnel service request with fifteen-bit length field.....	564
Table 342 – Coding rules for tunnel service response with seven-bit length field.....	564
Table 343 – Coding rules for tunnel service response with fifteen-bit length field .....	564
Table 344 – Coding rules for AlertReport service with seven bit length field .....	565
Table 345 – Coding rules for AlertReport service with fifteen-bit length field .....	565
Table 346 – Coding rules for AlertAcknowledge service .....	565
Table 347 – Coding rules for publish service for a native sequence of values .....	565
Table 348 – Coding rules for publish service – non-native (for tunnel support) .....	566
Table 349 – Coding rules for concatenate service .....	566
Table 350 – General coding rule for size-invariant application data .....	566
Table 351 – Coding rules for application data of varying size .....	566
Table 352 – Coding rules for Boolean data – true .....	567
Table 353 – Coding rules for Boolean data – false .....	567
Table 354 – Coding rules for Unsigned8 .....	568
Table 355 – Coding rules for Unsigned16 .....	568
Table 356 – Coding rules for Unsigned32 .....	568
Table 357 – Coding rules for Unsigned64 .....	569
Table 358 – Coding rules for Unsigned128.....	569
Table 359 – Coding rules for Float .....	570
Table 360 – Coding rules for double-precision float.....	570
Table 361 – Coding rules for VisibleString .....	571
Table 362 – Coding rules for OctetString .....	571
Table 363 – Coding rules for Bitstring .....	571
Table 364 – Example of coding for Bitstring of size 8 .....	571
Table 365 – Coding rules for TAITimeDifference .....	572
Table 366 – Coding rules for TAINetworkTimeValue .....	572
Table 367 – Coding rules for TAITimeRounded .....	573
Table 368 – Coding example: Read request for a non-indexed attribute .....	587
Table 369 – Coding example: Read response (corresponding to request contained in the preceding table) .....	588
Table 370 – Coding example: Tunnel service request .....	588
Table 371 – Summary of gateway high side interface services .....	594
Table 372 – Primitive G_Session parameter usage .....	603
Table 373 – GS_Status for G_Session confirm .....	604

Table 374 – Primitive G_Lease parameter usage .....	604
Table 375 – GS_Lease_Type for G_Lease request .....	605
Table 376 – GS_Status for G_Lease confirm .....	606
Table 377 – Primitive G_Device_List_Report parameter usage .....	607
Table 378 – GS_Status for G_Device_List_Report confirm.....	608
Table 379 – Primitive G_Topology_Report parameter usage .....	608
Table 380 – GS_Status for G_Topology_Report confirm .....	609
Table 381 – Primitive G_Schedule_Report parameter usage.....	609
Table 382 – GS_Status for G_Schedule_Report confirm .....	611
Table 383 – Primitive G_Device_Health_Report parameter usage.....	611
Table 384 – GS_Status for G_Device_Health_Report confirm .....	612
Table 385 – Primitive G_Neighbor_Health_Report parameter usage .....	612
Table 386 – GS_Status for G_Device_Health_Report confirm .....	613
Table 387 – Primitive G_Network_Health_Report parameter usage .....	614
Table 388 – GS_Status for G_Network_Health_Report confirm .....	615
Table 389 – Primitive G_Time parameter usage .....	616
Table 390 – GS_Status for G_Time confirm .....	616
Table 391 – Primitive G_Client_Server parameter usage .....	617
Table 392 – GS_Status for G_Client_Server confirm .....	618
Table 393 – Primitive G_Publish parameter usage .....	619
Table 394 – GS_Status for G_Publish confirm .....	620
Table 395 – Primitive G_Subscribe parameter usage .....	620
Table 396 – GS_Status for G_Subscribe confirm .....	621
Table 397 – Primitive G_Publish_Timer parameter usage .....	621
Table 398 – Primitive G_Subscribe_Timer parameter usage .....	621
Table 399 – Primitive G_Publish_Watchdog parameter usage .....	622
Table 400 – Primitive G_Bulk_Open parameter usage .....	623
Table 401 – GS_Status for G_Bulk_Open confirm .....	623
Table 402 – Primitive G_Bulk_Transfer parameter usage .....	624
Table 403 – GS_Status for G_Bulk_Transfer confirm .....	624
Table 404 – Primitive G_Bulk_Close parameter usage .....	624
Table 405 – GS_Status for G_Bulk_Close confirm .....	625
Table 406 – Primitive G_Alert_Subscription parameter usage .....	625
Table 407 – GS_Status for G_Alert_Subscription confirm .....	626
Table 408 – Primitive G_Alert_Notification parameter usage .....	627
Table 409 – Primitive G_Read_Gateway_Configuration parameter usage .....	628
Table 410 – GS_Attribute_Identifier values .....	628
Table 411 – GS_Status for G_Read_Gateway_Configuration confirm.....	629
Table 412 – Primitive G_Write_Gateway_Configuration parameter usage .....	629
Table 413 – GS_Attribute_Identifier values .....	629
Table 414 – GS_Status for G_Write_Gateway_Configuration confirm.....	630
Table 415 – Primitive G_Write_Device_Configuration parameter usage .....	630
Table 416 – GS_Status for G_Write_Device_Configuration confirm.....	631

Table 417 – Primitive G_Read_Device_Configuration parameter usage .....	631
Table 418 – GS_Status for G_Read_Device_Configuration confirm .....	632
Table 419 – UAP management object extended attributes .....	650
Table 420 – Factory default settings .....	658
Table 421 – Device provisioning object .....	664
Table 422 – Reset_To_Default method .....	668
Table 423 – Write symmetric join key method .....	668
Table 424 – Device provisioning service object .....	670
Table 425 – DPSOWhiteListTbl data structure .....	674
Table 426 – Array manipulation table .....	675
Table 427 – DPSO alert to indicate join by a device not on the WhiteList .....	676
Table 428 – DPSO alert to indicate inadequate device join capability .....	676
Table 429 – Field media type .....	682
Table 430 – Protocol layer support.....	683
Table 431 – Device PICS .....	683
Table 432 – PICS for device implementing I/O role .....	683
Table 433 – PICS for device implementing router role .....	684
Table 434 – PICS for backbone router .....	684
Table 435 – PICS for gateway.....	684
Table 436 – PICS for system manager .....	684
Table 437 – PICS for provisioning device .....	684
Table 438 – PICS for security manager .....	685
Table 439 – PICS for device implementing system time source role .....	685
Table 440 – System PICS .....	685
Table 441 – Device PICS .....	686
Table 442 – Router and backbone router PICS .....	686
Table 443 – PICS for system time source .....	687
Table 444 – PICS for system manager .....	687
Table 445 – PICS for provisioning role .....	688
Table 446 – Device PICS .....	688
Table 447 – PICS for provisioning role .....	689
Table 448 – PICS for security manager .....	689
Table 449 – PhL roles .....	690
Table 450 – PhL frequency of operation .....	690
Table 451 – PhL functions .....	690
Table 452 – PhL packet .....	691
Table 453 – DL roles .....	691
Table 454 – DL PICS for device implementing I/O role .....	691
Table 455 – DL PICS for device implementing router role .....	691
Table 456 – DL PICS for device implementing backbone router role .....	692
Table 457 – PICS for devices implementing I/O role .....	692
Table 458 – PICS for device implementing router role .....	693
Table 459 – PICS for devices implementing backbone router role .....	693

Table 460 – PICS for device implementing I/O role .....	694
Table 461 – PICS for routing device.....	694
Table 462 – PICS for backbone router .....	694
Table 463 – AL implementation option .....	695
Table 464 – PICS part 2: Optional industry-independent objects .....	695
Table 465 – PICS part 2: Supported standard services for I/O device role.....	696
Table 466 – PICS part 2: Supported standard services for system manager role .....	697
Table 467 – PICS part 2: Supported standard services for gateway role when supporting native access.....	698
Table 468 – PICS part 2: Supported standard services for gateway role when supporting interoperable tunneling and for adapters .....	699
Table 469 – PICS part 2: Supported standard services for routing device role .....	700
Table 470 – PICS part 2: Supported standard services for backbone router role .....	701
Table 471 – PICS part 2: Supported standard services for provisioning role .....	702
Table 472 – PICS part 2: Supported standard services for system time source role .....	702
Table 473 – Process control conformance: Supported objects .....	702
Table 474 – Process control conformance: Supported alerts .....	703
Table 475 – PICS: Gateway .....	704
Table 476 – PICS: I/O devices, routing devices, gateways, and backbone routers .....	706
Table 477 – PICS: Provisioning devices.....	706
Table 478 – Protocol layer device roles .....	708
Table 479 – Over-the-air upgrades.....	708
Table 480 – Session support profiles .....	709
Table 481 – Baseline profiles .....	709
Table 482 – PhL roles .....	710
Table 483 – DL required for listed roles .....	710
Table 484 – Role profiles: General DLMO attributes .....	711
Table 485 – Role profiles: dlmo.Device_Capability .....	712
Table 486 – Role profiles: dlmo.Ch (channel hopping) .....	713
Table 487 – Role profiles: dlmo.TsTemplate .....	713
Table 488 – Role profiles: dlmo.Neighbor.....	713
Table 489 – Role profiles: dlmo. NeighborDiag .....	714
Table 490 – Role profiles: dlmo.Superframe.....	714
Table 491 – Role profiles: dlmo.Graph .....	714
Table 492 – Role profiles: dlmo.Link .....	715
Table 493 – Role profiles: dlmo.Route .....	715
Table 494 – Role profiles: dlmo.Queue_Priority .....	715
Table 495 – Routing table size .....	715
Table 496 – Address table size .....	716
Table 497 – Port support size .....	716
Table 498 – APs .....	716
Table 499 – Role profile: Gateway .....	716
Table 500 – Role profile: Gateway native access .....	717
Table 501 – Role profile: Gateway interoperable tunnel mechanism .....	717

Table 502 – Role profiles: I/O, routers, gateways, and backbone routers .....	717
Table 503 – Usage classes .....	718
Table 504 – System management configuration defaults .....	723
Table 505 – Security configuration defaults .....	724
Table 506 – DL configuration defaults .....	725
Table 507 – Network configuration defaults .....	726
Table 508 – Transport configuration defaults .....	726
Table 509 – Application configuration defaults .....	727
Table 510 – Gateway configuration defaults .....	728
Table 511 – Provisioning configuration defaults .....	729
Table 512 – Table of standard object types .....	743
Table 513 – Template for standard object attributes .....	743
Table 514 – Template for standard object methods .....	744
Table 515 – Template for standard object alert reporting .....	745
Table 516 – Template for data structures .....	746
Table 517 – Scheduled_Write method template .....	747
Table 518 – Read_Row method template .....	748
Table 519 – Write_Row method template .....	748
Table 520 – Reset_Row method template .....	749
Table 521 – Delete_Row method template .....	750
Table 522 – Standard object types .....	753
Table 523 – Standard object instances .....	755
Table 524 – Standard data types .....	758
Table 525 – Protocol identification values .....	759
Table 526 – Sample MHR for join request .....	786
Table 527 – Sample DHR for join request .....	786
Table 528 – Network header for join messages .....	787

Figure 1 – Standard-compliant network .....	62
Figure 2 – Single protocol data unit .....	63
Figure 3 – Full protocol data unit .....	63
Figure 4 – Physical devices versus roles .....	70
Figure 5 – Notional representation of device phases .....	73
Figure 6 – Simple star topology .....	75
Figure 7 – Simple hub-and-spoke topology .....	76
Figure 8 – Mesh topology .....	77
Figure 9 – Simple star-mesh topology .....	77
Figure 10 – Network and DL subnet overlap .....	78
Figure 11 – Network and DL subnet differ .....	79
Figure 12 – Network with multiple gateways .....	80
Figure 13 – Basic network with backup gateway .....	81

Figure 14 – Network with backbone .....	82
Figure 15 – Network with backbone – device roles .....	83
Figure 16 – Reference model .....	84
Figure 17 – Basic data flow .....	85
Figure 18 – Data flow between I/O devices .....	86
Figure 19 – Data flow with legacy I/O device .....	87
Figure 20 – Data flow with backbone .....	87
Figure 21 – Data flow between I/O devices via backbone .....	88
Figure 22 – Data flow to standard-aware control system .....	89
Figure 23 – Management architecture .....	91
Figure 24 – DMAP .....	94
Figure 25 – Example of management SAP flow through standard protocol suite.....	96
Figure 26 – System manager architecture concept .....	115
Figure 27 – UAP-system manager interaction during contract establishment .....	134
Figure 28 – Contract-related interaction between DMO and SCO .....	136
Figure 29 – Contract source, destination, and intermediate devices .....	145
Figure 30 – Contract establishment example .....	153
Figure 31 – Contract ID usage in source .....	154
Figure 32 – Contract termination .....	157
Figure 33 – Contract modification with immediate effect .....	159
Figure 34 – Examples of DPDU and TPDU scope .....	161
Figure 35 – Keys and associated lifetimes .....	163
Figure 36 – Key lifetimes .....	164
Figure 37 – DPDU structure .....	167
Figure 38 – Outgoing messages – DL and security .....	168
Figure 39 – Incoming messages- DL and security .....	169
Figure 40 – TPDU structure and protected coverage .....	180
Figure 41 – TMIC parameters .....	180
Figure 42 – Transport layer and security sub-layer interaction, outgoing TPDU .....	182
Figure 43 – Transport layer and security sub-layer interaction, incoming TPDU .....	183
Figure 44 – Example: Overview of the symmetric key join process .....	198
Figure 45 – Example: Overview of the symmetric key join process of backbone device .....	199
Figure 46 – Asymmetric key-authenticated key agreement scheme .....	208
Figure 47 – Example: Overview of the asymmetric key join process for a device with a data link layer .....	211
Figure 48 – Example: Overview of the asymmetric key join process of a backbone device .....	212
Figure 49 – Device state transitions for join process and device lifetime .....	221
Figure 50 – High-level example of session establishment.....	222
Figure 51 – Key update protocol overview .....	230
Figure 52 – Device session establishment and key update state transition .....	236
Figure 53 – DL protocol suite and PPDU/DPDU structure .....	254
Figure 54 – Graph routing example .....	257
Figure 55 – Inbound and outbound graphs .....	259

Figure 56 – Slotted hopping .....	263
Figure 57 – Slow hopping .....	263
Figure 58 – Hybrid operation .....	264
Figure 59 – Radio spectrum usage .....	264
Figure 60 – Default hopping pattern 1 .....	266
Figure 61 – Two groups of devices with different hopping pattern offsets .....	267
Figure 62 – Interleaved hopping pattern 1 with 16 different hopping pattern offsets .....	268
Figure 63 – Slotted hopping .....	269
Figure 64 – Slow hopping .....	269
Figure 65 – Hybrid mode with slotted and slow hopping .....	270
Figure 66 – Combining slotted hopping and slow hopping .....	270
Figure 67 – Example of a three-timeslot superframe .....	271
Figure 68 – Superframes and links .....	271
Figure 69 – Multiple superframes, with timeslots aligned .....	272
Figure 70 – Slotted hopping .....	275
Figure 71 – Slow hopping .....	276
Figure 72 – Components of a slow hop .....	276
Figure 73 – Avoiding collisions among routers .....	277
Figure 74 – Hybrid configuration .....	278
Figure 75 – Timeslot allocation and the message queue .....	280
Figure 76 – 250 ms alignment intervals .....	283
Figure 77 – Timeslot durations and timing .....	283
Figure 78 – Clock source acknowledges receipt of DPDU .....	288
Figure 79 – Transaction timing attributes .....	290
Figure 80 – Dedicated and shared transaction timeslots .....	291
Figure 81 – Unicast transaction .....	292
Figure 82 – PDU wait time (PWT) .....	294
Figure 83 – Duocast support in the standard .....	295
Figure 84 – Duocast transaction .....	296
Figure 85 – Shared timeslots with CSMA-CA .....	297
Figure 86 – Transaction during slow-hopping periods .....	298
Figure 87 – DL management SAP flow through standard protocol suite .....	300
Figure 88 – PPDU and DPDU structure .....	317
Figure 89 – Typical acknowledgement frame layout .....	324
Figure 90 – Relationship among DLMO indexed attributes .....	356
Figure 91 – Address translation process .....	389
Figure 92 – Fragmentation process .....	390
Figure 93 – Reassembly process .....	392
Figure 94 – Processing of a NSDU received from the transport layer .....	394
Figure 95 – Processing of an incoming NPDU .....	395
Figure 96 – Processing of a NPDU received from the backbone .....	396
Figure 97 – Delivery of an incoming NPDU at its final destination .....	397
Figure 98 – Routing from a field device to a gateway on field – no backbone routing .....	397

Figure 99 – Protocol suite diagram for routing from a field device to a gateway on field – no backbone routing .....	398
Figure 100 – Routing a PDU from a field device to a gateway via a backbone router .....	399
Figure 101 – Protocol suite diagram for routing a PDU from a field device to a gateway via a backbone router .....	399
Figure 102 – Routing from a field device on one subnet to another field device on a different subnet.....	401
Figure 103 – Protocol suite diagram for routing from an I/O device on one subnet to another I/O device on a different subnet .....	402
Figure 104 – Routing over an Ethernet backbone network.....	403
Figure 105 – Routing over a fieldbus backbone network.....	403
Figure 106 – Distinguishing between NPDU header formats.....	414
Figure 107 – Transport layer reference model.....	421
Figure 108 – UDP “pseudo-header” for IPv6 .....	423
Figure 109 – Transport layer protocol data unit .....	426
Figure 110 – User application objects in a user application process .....	440
Figure 111 – Alarm state model .....	445
Figure 112 – Event model .....	446
Figure 113 – A successful example of multiple outstanding requests, with response concatenation .....	451
Figure 114 – An example of multiple outstanding unordered requests, with second write request initially unsuccessful .....	452
Figure 115 – An example of multiple outstanding ordered requests, with second write request initially unsuccessful .....	453
Figure 116 – Send window example 1, with current send window smaller than maximum send window .....	455
Figure 117 – Send window example 2, with current send window the same size as maximum send window, and non-zero usable send window width .....	455
Figure 118 – Send window example 3, with current send window the same size as maximum send window, and usable send window width of zero (0) .....	456
Figure 119 – General addressing model.....	458
Figure 120 – UAP management object state diagram .....	464
Figure 121 – Alert report reception state diagram .....	466
Figure 122 – Alert reporting example .....	466
Figure 123 – Upload/Download object download state diagram .....	480
Figure 124 – Upload/Download object upload state diagram .....	482
Figure 125 – Publish sequence of service primitives .....	500
Figure 126 – Client/server model two-part interactions .....	504
Figure 127 – Client/server model four-part interactions: Successful delivery .....	505
Figure 128 – Client/server model four-part interactions: Request delivery failure .....	505
Figure 129 – Client/server model four-part interactions: Response delivery failure .....	506
Figure 130 – AlertReport and AlertAcknowledge, delivery success.....	519
Figure 131 – AlertReport, delivery failure .....	519
Figure 132 – Alert Report, acknowledgment failure .....	520
Figure 133 – Concatenated response for multiple outstanding write requests (no message loss).....	526

Figure 134 – Management and handling of malformed APDUs received from device X .....	532
Figure 135 – Gateway scenarios .....	591
Figure 136 – Basic gateway model.....	592
Figure 137 – Sequence of primitives for session service .....	595
Figure 138 – Sequence of primitives for lease management service.....	595
Figure 139 – Sequence of primitives for system report services .....	596
Figure 140 – Sequence of primitives for time service .....	596
Figure 141 – Sequence of primitives for client/server service initiated from gateway .....	597
Figure 142 – Sequence of primitives for publish service initiated from gateway .....	597
Figure 143 – Sequence of primitives for subscribe service initiated from device.....	598
Figure 144 – Sequence of primitives for publisher timer initiated from gateway .....	598
Figure 145 – Sequence of primitives for subscriber timers initiated from device .....	598
Figure 146 – Sequence of primitives for the bulk transfer service .....	599
Figure 147 – Sequence of primitives for the alert subscription service .....	599
Figure 148 – Sequence of primitives for the alert notification service .....	600
Figure 149 – Sequence of primitives for gateway management services .....	600
Figure 150 – Tunnel object model .....	633
Figure 151 – Distributed tunnel endpoints .....	634
Figure 152 – Multicast, broadcast, and one-to-many messaging .....	635
Figure 153 – Tunnel object buffering .....	636
Figure 154 – Publish/subscribe publisher CoSt flowchart .....	638
Figure 155 – Publish/subscribe publisher periodic flowchart.....	639
Figure 156 – Publish/subscribe subscriber common periodic and CoSt flowchart .....	639
Figure 157 – Network address mappings .....	640
Figure 158 – Connection_Info usage in protocol translation .....	641
Figure 159 – Transaction_Info usage in protocol translation.....	642
Figure 160 – Interoperable tunneling mechanism overview diagram .....	643
Figure 161 – Bulk transfer model .....	645
Figure 162 – Alert model.....	646
Figure 163 – Alert cascading.....	647
Figure 164 – Native P/S and C/S access .....	648
Figure 165 – The provisioning network.....	655
Figure 166 – State transition diagrams outlining provisioning steps during a device life cycle .....	657
Figure 167 – State transition diagram showing various paths to joining a secured network .....	660
Figure 168 – Provisioning objects and interactions.....	662
Figure 169 – Basic reference model.....	720
Figure 170 – Generic protocol translation publish diagram .....	762
Figure 171 – Generic protocol translation subscribe diagram .....	763
Figure 172 – Generic protocol translation client/server transmission diagram.....	764
Figure 173 – Generic protocol translation client/server reception diagram .....	765
Figure 174 – Host integration reference model.....	778
Figure 175 – Configuration using an electronic device definition .....	780

Figure 176 – Configuration using FDT/DTM approach ..... 780

## INTRODUCTION

### 0.1 General

The ISA100 Committee was established by ISA to address wireless manufacturing and control systems in areas including:

- The environment in which the wireless technology is deployed;
- Technology and life cycle for wireless equipment and systems; and
- The application of wireless technology.

The Committee's focus is to improve the confidence in, integrity of, and availability of components or systems used for manufacturing or control, and to provide criteria for procuring and implementing wireless technology in the control system environment. Compliance with the Committee's guidance will improve manufacturing and control system deployment and will help identify vulnerabilities and address them, thereby reducing the risk of compromising or causing manufacturing control systems degradation or failure.

This ISA standard is intended to provide reliable and secure wireless operation for non-critical monitoring, alerting, supervisory control, open loop control, and closed loop control applications. This standard defines the protocol suite, system management, gateway, and security specifications for low-data-rate wireless connectivity with fixed, portable, and moving devices supporting very limited power consumption requirements. The application focus is to address the performance needs of applications such as monitoring and process control where latencies on the order of 100 ms can be tolerated, with optional behavior for shorter latency.

To meet the needs of industrial wireless users and operators, this standard provides robustness in the presence of interference found in harsh industrial environments and with legacy non-ISA100 compliant wireless systems. As described in Clause 4, this standard addresses coexistence with other wireless devices anticipated in the industrial workspace, such as cell phones and devices based on IEEE 802.11x, IEEE 802.15x, IEEE 802.16x, and other relevant standards. Furthermore, this standard allows for interoperability of ISA100 devices, as described in Clause 5.

This standard does not define or specify plant infrastructure or its security or performance characteristics. However, it is important that the security of the plant infrastructure be assured by the end user.

### 0.2 Document structure

This standard is organized into clauses focused on unique network functions and protocol suite layers. The clauses describe system, system management, security management, physical layer, data link layer, network layer, transport layer, application layer, gateway, and provisioning. Each clause describes a functionality or protocol layer and dictates the behavior required for proper operation. When a clause describes behaviors related to another function or layer, a reference to the appropriate clause will be supplied for further information.

The following terms will be used in this document to describe device behavior:

- Mandatory: behavior or a protocol that is required for a device to state compliance with the standard (e.g., symmetrical keys).
- Optional: behavior or protocol defined in the standard that is not required for compliance to the standard but, if implemented, shall be compliant with the standard (e.g., asymmetrical keys).
- Configuration: setting of a parameter that will alter behavior and can be set by the system manager (e.g., network layer hop limit). Configurations where defaults are appropriate will state those defaults (e.g., network layer hop limit = 64).
- Capability: ability of device to perform to a specified level (e.g., number of children a router can support). Profiles will specify a minimum capability.
- Feature: notable characteristic of a device (e.g., battery powered).

Mandatory behavior (also referred to as normative behavior) is denoted by the use of the term "shall". Non-mandatory behavior (also referred to as informative behavior) is denoted by the use of the terms "may" or "recommended".

The mandatory and optional communication protocols defined by this standard are referred to as native protocols, while those protocols used by other networks such as legacy fieldbus or HART communication protocols are referred to as foreign protocols.

## 0.2 Disclaimers

ISA does not take any position with respect to the existence or validity of any patent rights asserted in connection with this document, and ISA disclaims liability for the infringement of any patent resulting from the use of this document. Users are advised that determination of the validity of any patent rights, and the risk of infringement of such rights, is entirely their own responsibility.

Pursuant to ISA's Patent Policy, one or more patent holders or patent applicants may have disclosed patents that could be infringed by use of this document and executed a Letter of Assurance committing to the granting of a license on a worldwide, non-discriminatory basis, with a fair and reasonable royalty rate and fair and reasonable terms and conditions. For more information on such disclosures and Letters of Assurance, contact ISA or visit [www.isa.org/StandardsPatents](http://www.isa.org/StandardsPatents).

Other patents or patent claims may exist for which a disclosure or Letter of Assurance has not been received. ISA is not responsible for identifying patents or patent applications for which a license may be required, for conducting inquiries into the legal validity or scope of patents, or determining whether any licensing terms or conditions provided in connection with submission of a Letter of Assurance, if any, or in any licensing agreements are reasonable or non-discriminatory.

ISA requests that anyone reviewing this Document who is aware of any patents that may impact implementation of the Document notify the ISA Standards and Practices Department of the patent and its owner.

Additionally, the use of this standard may involve hazardous materials, operations or equipment. The standard cannot anticipate all possible applications or address all possible safety issues associated with use in hazardous conditions. The user of this standard must exercise sound professional judgment concerning its use and applicability under the user's particular circumstances. The user must also consider the applicability of any governmental regulatory limitations and established safety and health practices before implementing this standard.

NOTE The ISA100 standards development committee, which developed this ISA standard, is seeking feedback on its content and usefulness. If you have comments on the value of this document or suggestions for improvements or additional topics, please send those comments by email, fax, post, or phone to:

ISA100  
ISA Standards  
67 Alexander Drive  
Research Triangle Park, NC 27709 USA  
Email: [standards@isa.org](mailto:standards@isa.org)  
Tel: +1 919 990 9200 Fax: +1 919 549 8288

## REVISION HISTORY

ISA100.11a-2009		First approved version
ISA100.11a-rev		Revision of first approved version

ISA

---

**ISA100.11a****Wireless systems for industrial automation:  
Process control and related applications****1 Scope**

This project will define the OSI layer specifications (e.g., PhL, DL, etc.), security specifications, and management (including network and device configuration) specifications for wireless devices serving application classes 1 through 5 and optionally class 0 for fixed, portable, and moving devices.

NOTE Usage classes are described in Annex C.

The project's application focus will address performance needs for periodic monitoring and process control where latencies on the order of 100 ms can be tolerated, with optional behavior for shorter latency.

This project will address:

- Low energy consumption devices, and also those applications that have latency and latency variability constraints, with the ability to scale to address large installations;
- Wireless infrastructure, interfaces to legacy infrastructure and applications, security, and network management requirements in a functionally scalable manner;
- Robustness in the presence of interference found in harsh industrial environments and with legacy systems;
- Coexistence with other wireless devices anticipated in the industrial work space, such as IEEE 802.11x, IEEE 802.16x, cell phones, and other relevant standards; and
- Interoperability of ISA100 devices.

**2 Normative references**

The following standards and specifications contain provisions which, through references in this text, constitute provisions of this standard. At the time of publication, the editions indicated were valid. All standards and specifications are subject to revision and may be changed without notice, and parties to agreements based on this standard are encouraged to investigate the possibility of applying the most recent editions of the references listed below.

For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE See the bibliography for non-normative references.

ANSI X9.63-2001, *Public key cryptography for the financial services industry - Key agreement and key transport using elliptic curve cryptography*. American Bankers Association, November 20, 2001

FIPS 197, *Advanced encryption standard (AES)*, Federal Information Processing Standards Publication 197, US Department of Commerce/N.I.S.T., Springfield, Virginia, November 26, 2001

FIPS 198, *The keyed-hash message authentication code (HMAC)*, Federal Information Processing Standards Publication 198, US Department of Commerce/N.I.S.T., Springfield, Virginia, March 6, 2002

IEEE Std 802.15.4™:2006, *Wireless medium access control (MAC) and physical layer (PhL) specifications for low rate wireless personal area networks (WPANs)*

IETF RFC 4944, *Transmission of IPv6 packets over IEEE Std 802.15.4 networks*  
<ftp://ftp.rfc-editor.org/in-notes/internet-drafts/draft-ietf-6LoWPAN-format-13.txt>

ISO/IEC 646:1991 ISO 7-Bit Coded Character Set for Information Interchange (Formerly known as ISO/IEC 646-1991)

ISO/IEC 7498-1:1994, *Information technology – Open systems interconnection – Basic reference model: The basic model*

ISO 7498-2:1989, *Information processing systems – Open systems interconnection – Basic reference model – Part 2: Security architecture*

ISO/IEC 7498-3:1997, *Information technology – Open systems interconnection – Basic reference model: Naming and addressing*

ISO/IEC 7498-4:1989, *Information processing systems – Open systems interconnection – Basic reference model – Part 4: Management framework*

ISO/IEC 10118-2, *Information technology – Security techniques – Hash-functions – Part 2: Hash-functions using an n-bit block cipher*

ISO/IEC 10731, *Information technology – Open systems interconnection – Basic reference model – Conventions for the definition of OSI services*

NIST SP 800-38C:2004, *Recommendation for block cipher modes of operation – The CCM mode for authentication confidentiality*

NIST SP 800-56a:2007, *Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography*

NIST SP 800-57:2007, *Recommendation for key management – Part 1: General*

NIST SP 800-57:2005, *Recommendation for Key Management – Part 2: Best practices for key management organization*

*Standards for efficient cryptography*, SEC 1: Elliptic curve cryptography, version 1.0, Certicom Research, September 20, 2000

*Standards for efficient cryptography*, SEC 2: Recommended elliptic curve domain parameters, version 1.0, Certicom Research, September 20, 2000

*Standards for efficient cryptography*, SEC 4: Elliptic curve Qu-Vanstone implicit certificate scheme, version 0.9, Certicom Research, November 14, 2007

### **3 Terms, definitions, abbreviated terms, acronyms, and conventions**

#### **3.1 (N)-layer and other terms and definitions from the open systems interconnection basic reference model**

NOTE All references are with respect to ISO/IEC 7498-1:1994 as corrected and reprinted in 1996, unless otherwise identified.

##### **3.1.1 acknowledgement [4.2.3.4, 5.8.1.16]**

function of the (N)-layer that allows a receiving (N)-layer entity to inform a sending (N)-layer entity of the receipt of a PDU

##### **3.1.2 application-entity, AE [7.1.1.1]**

active element, within an application process, embodying a set of capabilities that is pertinent to OSI and that is defined for the application layer, that corresponds to a specific application-entity-type (without any extra capabilities being used)

NOTE This is a slight specialization of (N)-entity, because the application layer includes non-OSI-relevant application functions. Each application-entity represents one and only one process in the open system interconnection environment.

**3.1.3****application-management [8.1.1]**

functions in the application layer related to management of OSI application-processes

**3.1.4****(N)-association [15.5.3.2]**

cooperative relationship among (N)-layer entity invocations

**3.1.5****blocking [5.8.1.11]**

function performed by an (N)-entity to map multiple (N)-SDUs into one (N)-PDU

**3.1.6****centralized-multi-endpoint-connection [5.8.1.2]**

multi-endpoint-connection where data sent by the entity associated with the central-connection-endpoint is received by all other entities, while data sent by the other entities is received by only the central entity

**3.1.7****concatenation [5.8.1.13]**

function performed by an (N)-entity to map multiple (N)-PDUs into one (N-1)-SDU

NOTE Blocking and concatenation, though similar (they both permit grouping of data-units) may serve different purposes. For instance, concatenation permits the (N)-layer to group one or several acknowledgement (N)-PDUs with one (or several) (N)-PDUs containing user-data. This would not be possible with the blocking function only. Note also that the two functions may be combined so that the (N)-layer performs blocking and concatenation.

**3.1.8****(N)-connection [5.2.1.1]**

association requested by an (N+1) layer entity for the transfer of data between two or more (N+1) entities

**3.1.9****(N)-connection endpoint [14.4.7]**

terminator at one end of an (N)-connection within an (N)-service-access-point

**3.1.10****(N)-connection-endpoint-identifier [5.4.1.5]**

identifier of an (N)-connection-endpoint which can be used to identify the corresponding (N)-connection at an (N)-SAP

**3.1.11****(N)-connection-endpoint-suffix [5.4.1.6]**

that part of an (N)-connection-endpoint-identifier which is unique within the scope of an (N)-SAP

**3.1.12****(N)-connection-mode-transmission [5.3.1.17]**

(N)-data-transmission in the context of an (N)-connection

**3.1.13****(N)-connectionless-mode-transmission [5.3.1.18]**

(N)-data-transmission not in the context of an (N)-connection and not required to maintain any logical relationship between (N)-SDUs

**3.1.14****correspondent-(N)-entities [5.3.1.5]**

(N)-entities with an (N-1)-connection between them

**3.1.15****(N)-data transmission [5.3.3.2]**

(N)-facility that conveys SDUs from one (N+1) layer entity to one or more (N+1) entities

**3.1.16****deblocking [5.8.1.12]**

function performed by an (N)-entity to identify multiple (N)-SDUs which are contained in one (N)-PDU

NOTE In the absence of error, deblocking is the reverse function of blocking.

**3.1.17****decentralized-multi-endpoint-connection [5.8.1.2]**

multi-endpoint-connection where data sent by an entity associated with a connection-endpoint is received by all other entities

**3.1.18****demultiplexing [5.8.1.5]**

function performed by an (N)-entity which identifies (N)-PDUs for more than one (N)-connection within an (N-1)-connection

NOTE In the absence of error, demultiplexing is the reverse function of multiplexing.

**3.1.19****(N)-entity [5.2.1.11]**

active element within an (N)-subsystem embodying a set of capabilities defined for the (N)-layer that corresponds to a specific (N)-entity-type (without any extra capabilities being used)

**3.1.20****(N)-entity-invocation [5.2.1.12]**

specific utilization of part or all or all of the capabilities of a given (N)-entity (without any extra capabilities being used)

**3.1.21****(N)-entity-type [5.2.1.10]**

description of a class of (N)-entities in terms of a set of capabilities defined for the (N)-layer

**3.1.22****(N)-interface-control-information, (N)-ICI [5.6.1.4]**

information transferred between an (N+1)-entity and an (N)-entity to coordinate their joint operation

**3.1.23****(N)-layer [4.2.1]**

subdivision of the OSI architecture, constituted by subsystems of the same rank (N)

**3.1.24****(N)-layer-management [8.1.6]**

functions related to the management of the (N)-layer partly performed in the (N)-layer itself according to the (N)-protocol of the layer (activities such as activation and error control) and partly performed as a subset of systems-management

**3.1.25****multi-endpoint-connection [5.3.1.4]**

connection with more than two connection-endpoints

**3.1.26****multiplexing [5.8.1.4]**

function performed by an (N)-entity in which one (N-1)-connection is used to support more than one (N)-connection

NOTE The term multiplexing is also used in a more restricted sense to the function performed by the sending (N)-entity while the term demultiplexing is used to the function performed by the receiving (N)-entity.

**3.1.27****peer-(N)-entities [5.2.1.3]**

entities within the same (N)-layer

**3.1.28****(N)-protocol [5.3.6]**

set of rules and formats (semantic and syntactic) that determines the communication behavior of (N)-entities in the performance of (N)-functions

**3.1.29****(N)-protocol-addressing-information, (N)-PAI [ISO/IEC 7498-3:1997, 3.4.20]**

those elements of (N)-PCI which contain addressing information

**3.1.30****(N)-protocol-control-information, (N)-PCI [5.6.1.1]**

information exchanged between (N)-entities to coordinate their joint operation

**3.1.31****(N)-protocol-data-unit, (N)-PDU [5.6.1.3]**

unit of data specified in an (N)-protocol and consisting of (N)-protocol-control-information and possibly (N)-user-data

**3.1.32****(N)-protocol-version-identifier [5.8.1.18]**

identifier conveyed between correspondent (N)-entities which allows the selection of the version of an (N)-protocol

**3.1.33****quality of service, QoS [4.2.2.2]**

collective effect of service performance which determines the degree of satisfaction of a user of the service; the negotiated parameters for a link including:

- Priority;
- Time windows for control messaging;
- Acceptability of out-of-order message delivery; and
- Acceptability of message delivery in partial increments.

NOTE 1 The quality of service is characterized by the combined aspects of service support performance, service operability performance, serveability performance, service integrity and other factors specific to each service.

NOTE 2 ISO defines quality as the ability of a product or service to satisfy users' needs.

**3.1.34****reassembling [5.8.1.10]**

function performed by an (N)-entity to map multiple (N)-PDUs into one (N)-SDU

NOTE In the absence of error, reassembling is the reverse function of segmenting.

**3.1.35****recombining [5.8.1.7]**

function performed by an (N)-entity which identifies (N)-PDUs for a single (N)-connection in (N-1)-SDUs received on more than one (N-1)-connection

NOTE In the absence of error, recombining is the reverse function of splitting.

**3.1.36****(N)-relay [5.3.1.6]**

(N)-function by means of which an (N)-entity forwards data received from one peer (N)-entity to another peer (N)-entity

**3.1.37****reset [5.8.1.17]**

function that sets the corresponding (N)-entities to a predefined state with a possible loss or duplication of data

**3.1.38****segmenting [5.8.1.9]**

function performed by an (N)-entity to map one (N)-SDU into multiple (N)-PDUs

**3.1.39****(N)-selector [derived from ISO/IEC 7498-3:1997, 6.2.3]**

that part of an (N)-address that is specific to the addressed (N)-subsystem, i.e., which identifies one or more (N)-SAPs within an end open system once that end open system is unambiguously addressed

**3.1.40****separation [5.8.1.14]**

function performed by an (N)-entity to identify multiple (N)-PDUs which are contained in one (N-1)-SDU

NOTE In the absence of error, separation is the reverse function of concatenation.

**3.1.41****sequencing [5.8.1.15]**

function performed by the (N)-layer to preserve the order of (N)-SDUs that were submitted to the (N)-layer

**3.1.42****(N)-service [4.2.2]**

capability of the (N)-layer and the layers beneath it, which is provided to (N+1) entities at the boundary between the (N)-layer and the (N+1) layer

**3.1.43****(N)-service access point, (N)-SAP [7.3.3]**

point at which (N)-services are provided by an (N)-layer entity to an (N+1) layer entity

**3.1.44****(N)-service-access-point-address, (N)-SAP-address [5.4.1.2]**

(N)-address that is used to identify a single (N)-SAP

**3.1.45****(N)-service-data-unit, (N)-SDU [5.6.1.4]**

amount of information whose identity is preserved when transferred between peer (N+1)-entities and which is not interpreted by the supporting (N)-entities

**3.1.46****splitting [5.8.1.6]**

function within the (N)-layer by which more than one (N-1)-connection is used to support one (N)-connection

NOTE The term splitting is also used in a more restricted sense to see the function performed by the sending (N)-entity while the term recombining is used to see the function performed by the receiving (N)-entity.

**3.1.47****system management [8.1.4]**

functions in the application layer related to management of various OSI resources and their status across all layers of the OSI architecture

**3.1.48****user application process (UAP) [4.2.2.1]**

active process within the highest portion of the application layer that is the user of OSI services

NOTE 1 The aspects of a UAP that need to be taken into account for the purpose of OSI are represented by one or more application-entities, of one or more application-entity-types. [7.1.2.2, 7.1.2.3]

NOTE 2 The collection of UAPs is sometimes referred to as the user layer, even though ISO/IEC 7498-1, 7.1.2.1 states that the application layer has no boundary with a higher layer. In the reference model, the application layer includes the UAPs.

**3.1.49****(N)-user-data [5.6.1.2]**

data transferred between (N)-entities on behalf of the (N+1)-entities for which the (N)-entities are providing services

## 3.2 Other terms and definitions

**3.2.1****access control**

mechanism that restricts access to resources to only privileged entities

**3.2.2****accountability**

property that ensures that the actions of an entity may be traced uniquely to that entity

**3.2.3****alert**

event, such as the occurrence of an alarm or a return-to-normal condition, of potential significance to a correspondent UAP

**3.2.4****application**

(verb) act of applying

(noun) system or problem to which a computer is applied; program that provides functionality to end users

(adjective) highest protocol layer in the ISO definition; types of applications according to criticality

**3.2.5****application process**

element that performs the information processing for a particular application

**3.2.6****association**

relationship for a particular purpose

**3.2.7****asymmetric key (cryptographic) algorithm**

public key cryptographic algorithm

**3.2.8****authentication**

process that establishes the origin of information, or determines an entity's identity

NOTE In a general information security context: Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system

**3.2.9****authentication code**

full or truncated cryptographic checksum based on an appropriate security function (see MIC, 3.2.94)

NOTE 1 This is also known as a message authentication code (MAC). It is called a message integrity code (MIC) when used in contexts where the acronym MAC has an alternate definition, such as in local area network standards.

**3.2.10****authorization**

access privileges that are granted to an entity; conveying an official sanction to perform a function or activity

**3.2.11****availability**

timely, reliable access to information by authorized entities

**3.2.12****backup**

copy of information to facilitate recovery, if necessary, during the cryptoperiod of a key

**3.2.13****bandwidth**

depending on the context of the citation:

- analog bandwidth (Hz); numerical difference between the upper and lower frequencies of a band of frequencies
- digital bandwidth (bits); amount of data that can be passed along a communications channel within a given period of time

**3.2.14****black channel**

communication channel of a safety system that provides no safety functionality in addition to its basic communication capability

**3.2.15****blacklist**

list of RF channels upon which transmission is prohibited

NOTE This may refer to a blacklist that is temporary, local, or network-wide, or loss of bandwidth.

**3.2.16****block cipher**

cryptographic primitive that uses a symmetric key to create a key-dependent pseudorandom permutation of a fixed-size bit string

**3.2.17****broadcast**

transmission intended for all nodes

NOTE 1 Broadcast reception may be limited to specific layers, i.e., MAC or network layer.

NOTE 2 Many lower layer protocols do not provide an acknowledgement for broadcasts.

### **3.2.18**

#### **cipher**

cryptographic algorithm for encryption and decryption [IETF RFC 2828]

### **3.2.19**

#### **ciphertext**

data that has been transformed by encryption so that its semantic information content (i.e., its meaning) is no longer intelligible or directly available (see cleartext, plaintext)

NOTE Relative to a PDU, ciphertext is information in a PDU that is subject to obscuration by encryption, in its post-encryption pre-decryption obscured form.

### **3.2.20**

#### **cleartext**

information in a PDU that is not subject to obscuration by encryption

NOTE Relative to a PDU, cleartext is information in the PDU that is not subject to obscuration by encryption, that when present in the PDU is always present in its unobscured form.

### **3.2.21**

#### **coexistence**

ability of multiple systems to perform their tasks in a given environment where they may or may not be using a similar set of rules

### **3.2.22**

#### **compromise**

unauthorized disclosure, modification, substitution, or use of sensitive data (e.g., keying material and other security-related information)

### **3.2.23**

#### **confidentiality**

property that sensitive information is not disclosed to unauthorized entities

NOTE In a general information security context, confidentiality preserves authorized restrictions on information access and disclosure, including means for preserving personal privacy and proprietary information.

### **3.2.24**

#### **construction option**

set of features that a device designer may choose to include in, or exclude from, a device

### **3.2.25**

#### **contract**

agreement between the system manager and a device in the network involving the allocation of network resources by the system manager to support a particular communication need of that device

### **3.2.26**

#### **cryptanalysis**

operations performed in defeating cryptographic protection without an initial knowledge of the key employed in providing the protection

study of mathematical techniques for attempting to defeat cryptographic techniques and information system security, including the process of looking for errors or weaknesses in an algorithm or in its implementation

### **3.2.27**

#### **cryptographic algorithm**

well-defined computational procedure that takes variable inputs including a cryptographic key and produces an output

NOTE Example cryptographic algorithms are block and stream ciphers and keyed hashes. An unkeyed hash is not formally a cryptographic algorithm, although it may be a one-way function that has similar resistance to attack, and may be constructed from a cryptographic algorithm with a fixed key. The SHA family of hashes is so constructed.

### 3.2.28

#### **cryptographic key (key)**

parameter used in conjunction with a cryptographic algorithm that determines its operation in such a way that an entity with knowledge of the key can reproduce or reverse the operation, while an entity without knowledge of the key cannot

NOTE Examples include

- transformation of plaintext data into ciphertext data
- transformation of ciphertext data into plaintext data
- computation of a digital signature from data
- authentication of a digital signature
- computation of an authentication code from data
- verification of an authentication code from data and a received authentication code
- computation of a shared secret that is used to derive keying material.

### 3.2.29

#### **cryptographic key component (key component)**

one of at least two parameters that have the same format as a cryptographic key; which are combined in an appropriate security function to form a plaintext cryptographic key before use

### 3.2.30

#### **cryptographic module**

set of hardware, software, and/or firmware that implements appropriate security functions (including cryptographic algorithms and key generation) and is contained within the cryptographic boundary

### 3.2.31

#### **cryptoperiod**

time span during which a specific key is authorized for use or in which the keys for a given system or application may remain in effect [from FIPS SP 800-57 Part 1]

### 3.2.32

#### **data authenticity**

property that sensitive data has not been modified in an unauthorized and undetected manner

### 3.2.33

#### **data integrity**

property that data has not been modified in an undetected manner by natural causes

### 3.2.34

#### **data key, data authenticating key, data encrypting key, DK**

cryptographic key that is used to cryptographically protect data (e.g., encrypt, decrypt, authenticate)

### 3.2.35

#### **decryption**

process of changing ciphertext into plaintext using a cryptographic algorithm and key

### 3.2.36

#### **deployment option**

set of features that a device designer shall include in a device, but which the end-user or his/her agent (e.g., a network security manager) can elect to employ or not employ

**3.2.37****derived key**

shared-secret symmetric key that is derived from a prior shared-secret symmetric key

NOTE Such keys can be used to limit the cryptoperiod of any single key while meeting key archive requirements, provided that the independent key from which the derived key was derived (perhaps through many generations of derivation) has previously met those archive requirements.

**3.2.38****destruction**

with regard to security, process of overwriting, erasing, or physically destroying a key so that it cannot be recovered

**3.2.39****device security management object**

application software within a device that acts as a local peer of a security manager

**3.2.40****digital signature**

result of a cryptographic transformation of data that, when properly implemented, provides the services of

- source authentication;
- data integrity;
- signer non-repudiation when the cryptographic transformation uses asymmetric-key cryptography.

**3.2.41****DL transaction**

any MPDU that is not an immediate acknowledgement MPDU, plus any immediate acknowledgement MPDUs that immediately follow the first MPDU and/or each other

**3.2.42****duocast**

variant of unicast, wherein a second receiver is scheduled to overhear the DPDU and provides a second acknowledgement

NOTE Duocast is shown graphically in Figure 84.

**3.2.43****encrypted key**

cryptographic key that has been encrypted using an appropriate security function with a key encrypting key in order to disguise the value of the underlying plaintext key

**3.2.44****encryption**

process of changing plaintext into ciphertext using a cryptographic algorithm and key

**3.2.45****entity**

individual (person), organization, device or process

**3.2.46****ephemeral key**

cryptographic key that is generated for each execution of a key establishment process and that meets other requirements of the key type (e.g., unique to each message or session)

NOTE In some cases ephemeral keys are used more than once, within a single session (e.g., broadcast applications) where the sender generates only one ephemeral key pair per message and the private key is combined separately with each recipient's public key.

**3.2.47  
field devices**

physical devices designed to meet the rigors of plant operation that communicate via DPDUs conforming to this standard

NOTE These include routing devices, sensors, and actuators.

**3.2.48  
field network**

configuration of two or more field devices interconnected by a protocol defined by this standard to transfer information

**3.2.49  
foreign protocol application communication (FPAC)**

optimized conveyance of PDUs or portions of PDUs from a first protocol within a second protocol by selective usage of caching, compression, address translation and proxy techniques

**3.2.50  
gateway**

device between communications compliant to this standard and other communications that acts as a protocol translator between the application layer of this standard and other application layers

**3.2.51  
hash-based message authentication code (HMAC)**

message authentication code that uses an appropriate keyed hash function

NOTE This definition is generalized from NIST SP 800-57 part 1, which refers to the construction specified in FIPS 198a.

**3.2.52  
hash function**

function that maps a bit string of arbitrary size to a fixed size bit string

NOTE 1 Appropriate hash functions satisfy the following two properties.

- (One-way) It is computationally infeasible to find any input that maps to any pre-specified output;
- (Collision resistant) It is computationally infeasible to find any two distinct inputs that map to the same output.

NOTE 2 See the note of 3.2.27.

**3.2.53  
hash value**

full or truncated result of applying a hash function to information

**3.2.54  
identity**

distinguishing character or personality of an individual or entity

**3.2.55  
independent key**

shared-secret symmetric key that is derived from a high entropy bit source and not from a prior key

**3.2.56  
infrastructure**

technical structures that support data communications within a facility; examples of infrastructure are a plant's IT network, perhaps placed on IEEE 802.3 or IEEE 802.11, PROFIBUS, or Fieldbus Foundation networks to field devices, etc.<sup>1</sup>

**3.2.57  
initialization vector, IV**

block of bits that is required to allow a cryptographic cipher in a streaming mode of operation to produce a unique stream independent from other streams produced under the same encryption key

**3.2.58  
interoperability**

ability of two or more devices to communicate using the communication protocol suite compliant with this standard

**3.2.59  
Kerberos protocol**

a specific network authentication protocol that allows individuals communicating over an insecure network to prove their identity to one another in a secure manner

**3.2.60  
key agreement**

key establishment procedure where resultant keying material is a function of information contributed by two or more participants, so that no party can predetermine the value of the keying material independent of the other party's contribution

**3.2.61  
key archive**

encryption system with a backup decryption capability that allows authorized persons (users, officers of an organization, and government officials), under certain prescribed conditions, to decrypt ciphertext with the help of information supplied by one or more trusted parties who hold special data recovery keys, which may but need not differ from the keys used to encrypt the information

**3.2.62  
key center**

centralized key distribution process, usually a separate computer system, that uses key-encrypting keys (master keys) to encrypt and distribute session keys needed by a community of users

NOTE Key centers generally need certification, traceable to an accredited independent testing agency, that they meet the requirements of FIPS 140-2 for a Level 3 or Level 4 cryptographic module.

**3.2.63  
key confirmation**

procedure to provide assurance to one party that another party actually possesses the same keying material and/or shared secret

---

<sup>1</sup> PROFIBUS and Fieldbus Foundation are the trademarks of various trade organizations. This information is given for the convenience of users of the standard and does not constitute an endorsement of the trademark holders or any of their products. Compliance to this profile does not require use of the registered trademark. Use of the trademarks requires permission of the trade name holder.

**3.2.64  
key de-registration**

marking of all keying material records and associations to indicate that the key is no longer in use

**3.2.65  
key derivation**

process by which one or more keys are derived from a shared secret and other information

**3.2.66  
key distribution**

transport of a key and other keying material from an entity that either owns the key or generates the key to another entity that is intended to use the key

**3.2.67  
key distribution center, KDC**

type of key center (used in symmetric cryptography) that implements a key distribution protocol to provide keys (usually, session keys) to two (or more) entities that wish to communicate securely

**3.2.68  
key encrypting key, KEK**

cryptographic key that is used for the encryption or decryption of other keys

NOTE Best practice limits use of symmetric (secret) KEKs to key wrapping and not key transport.

**3.2.69  
key escrow**

encryption system with a backup decryption capability that allows authorized persons (users, officers of an organization, and government officials), under certain prescribed conditions, to decrypt ciphertext with the help of information supplied by one or more trusted parties who hold special data recovery keys, which may (but are not required to) differ from the keys used to encrypt the information

**3.2.70  
key establishment**

process by which cryptographic keys are securely distributed among cryptographic modules using manual transport methods (e.g., key loaders), automated methods (e.g., key transport and/or key agreement protocols), or a combination of automated and manual methods (consisting of key transport plus key agreement)

**3.2.71  
keying material installation**

installation of keying material for operational use

**3.2.72  
key management**

activities involving the handling of cryptographic keys and other related security parameters (e.g., IVs and passwords) during the entire life cycle of the keys, including their generation, storage, establishment, entry and output, and destruction

**3.2.73  
key management archive**

repository containing keying material of historical interest

**3.2.74****key management infrastructure**

framework and services that provide for the generation, production, distribution, control, accounting, and destruction of all cryptographic material, including symmetric keys, as well as public key signing and generation of its own static and ephemeral asymmetric key pairs

NOTE 1 This includes all elements (hardware, software, other equipment, and documentation); facilities; personnel; procedures; standards; and information products that form the system that distributes, manages, and supports the delivery of cryptographic products and services to end users.

NOTE 2 Key management services include key ordering, distribution, re-key, update of keying material attributes, certificate revocation, key recovery and the distribution, accounting, tracking, and control of software that performs either keying material security or cryptographic functions.

**3.2.75****key manager**

key management infrastructure device that provides key management services

**3.2.76****(asymmetric-key) key pair**

public key and its corresponding private key

NOTE A key pair is used with asymmetric-key cryptographic algorithms.

**3.2.77****key recovery**

mechanisms and processes that allow authorized entities to retrieve keying material from key backup or archive

**3.2.78****key registration**

process of officially recording the keying material by a registration authority

**3.2.79****key revocation**

process whereby a notice is made available to affected entities that keying material should be removed from operational use prior to the end of the established cryptoperiod of that keying material

**3.2.80****key transport**

key establishment procedure whereby one party (the sender) selects and encrypts the keying material and then distributes the material to another party (the receiver)

NOTE When used in conjunction with a public key (asymmetric) algorithm, the keying material is encrypted using the public key of the receiver and subsequently decrypted using the private key of the receiver. When used in conjunction with a symmetric algorithm, the keying material is wrapped with a key encrypting key shared by the two parties.

**3.2.81****key update**

function performed on a cryptographic key in order to compute a new but related key

**3.2.82****key usage period**

either the originator usage period or the recipient usage period of a symmetric key

**3.2.83****key wrapping**

method of encrypting keys (along with associated integrity information) that provides both confidentiality and integrity protection using a symmetric key

**3.2.84****key wrapping key**

symmetric key encryption key

**3.2.85****keying material**

data (e.g., keys and IVs) necessary to establish and maintain cryptographic keying relationships

**3.2.86****label**

with regard to security, information associated with a key that identifies the key's parameters, attributes or intended use

**3.2.87****latency**

delay from when data is created at a data source device to when it is available to be consumed at the destination device)

NOTE The designated points may be a) physical devices, or b) layer boundaries within multi-layer software (e.g., from sending transport to receiving transport functionality, or from sending application to sending modem).

**3.2.88****lease**

gateway service access point per-session fine-grained communication resource allocation

**3.2.89****least privilege**

security principle that restricts the access privileges (e.g., program execution privileges, file modification privileges) of authorized personnel and their cyber agents to the minimum necessary to perform their jobs

**3.2.90****link**

momentary or persistent interconnecting path between two or more devices for the purpose of transmitting and receiving messaging

**3.2.91****master key, MK**

cryptographic key that is used for deriving other keys

NOTE Best practice prohibits MKs from being used as data keys, which could ease their cryptanalysis.

**3.2.92****mesh topology**

network topology in which redundant physically-diverse routing paths are available between each pair of network nodes

NOTE Wireless mesh topology may be used to extend coverage via multi-hop capability and may facilitate communication reliability by providing redundant paths between devices.

**3.2.93****message authentication**

process of establishing that a message was formed by a member of an authorized group of communicants and that the message is unchanged since it was formed

NOTE This is also known as PDU authentication.

**3.2.94****message integrity code, MIC**

usually-truncated message authentication code

**3.2.95****multicast**

messaging from a source to a set of intended recipients

NOTE 1 The set membership may be determinate or indeterminate, including null.

NOTE 2 Broadcast is a special form of multicast, usually to an indeterminate set of intended recipients. See also unicast.

NOTE 3 Multicast is not supported in this standard.

**3.2.96****network management object**

application software within a device that acts as a local peer of a network manager

**3.2.97****non-repudiation**

service that is used to provide assurance of the integrity and origin or delivery of data in such a way that the integrity and origin or delivery can be verified by a third party as having originated from or been delivered to a specific entity in possession of the private key of the claimed signatory

NOTE In a general information security context, non-repudiation provides assurance that the sender of information is provided with durable proof of delivery or the recipient is provided with durable proof of the sender's identity, so that the party that provided the non-repudiable proof cannot later deny having processed the information.

**3.2.98****nonce**

number used once, a value that has (at most) a negligible chance of repeating

**3.2.99****operational phase, operational use**

phase in the lifecycle of keying material whereby keying material is used for standard cryptographic purposes

**3.2.100****operational storage**

normal storage of operational keying material during its cryptoperiod

**3.2.101****originator usage period**

period of time during the cryptoperiod of a symmetric key during which cryptographic protection may be applied to data

**3.2.102****password**

string of characters (letters, numbers and other symbols, possibly including spaces and other word-delimiting characters) that are used to authenticate an identity or to verify access authorization

**3.2.103****PDU authentication**

see 3.2.93, message authentication

**3.2.104****period of protection**

period of time during which the integrity and/or confidentiality of a key needs to be maintained

**3.2.105  
plaintext**

data that is input to and transformed by an encryption process, or that is output by a decryption process

NOTE Usually, the plaintext input to an encryption operation is cleartext. But in some cases, the input is ciphertext that was output from another encryption operation.

**3.2.106  
private key**

cryptographic key, used with a public key cryptographic algorithm, that is uniquely associated with an entity and that is not made public

NOTE In an asymmetric (public) cryptosystem, the private key is associated with a public key. The private key is known only by the owner of the key pair and is used to:

- a) compute the corresponding public key;
- b) compute a digital signature that may be verified by the corresponding public key;
- c) decrypt data that was encrypted by the corresponding public key; or
- d) compute a piece of common shared data, together with other information.

**3.2.107  
pseudo-random bit generator**

process used to generate an unpredictable series of bits that are random in the sense that there is no way to describe the generators output that is more efficient than simply listing each entire output string

NOTE Pseudo-random bit generators have provable properties. The unpredictability of their output depends on the unpredictability of their initial input and the rate at which new unpredictable (high entropy) input is included relative to the number of output bits generated. See the note in 3.2.137, true-random bit generator, for common sources of such unpredictability.

**3.2.108  
public key**

cryptographic key used with a public key cryptographic algorithm that is uniquely associated with an entity and that may be made public

**3.2.109  
public key certificate**

set of data that uniquely identifies an entity, contains the entity's public key and possibly other information, and is digitally signed by a trusted party, thereby binding the public key to the entity

NOTE Additional information in the certificate could specify how the key is used and its cryptoperiod.

**3.2.110  
public key (asymmetric) cryptographic algorithm**

cryptographic algorithm that uses two related keys, a public key and a private key, where the two keys have the property that determining the private key from the public key is computationally infeasible

**3.2.111  
recipient usage period**

period of time during the cryptoperiod of a symmetric key during which the protected information is processed

NOTE This period frequently extends beyond the originator's period of permitted usage.

**3.2.112  
resilience**

trait describing a system's ability to accommodate significant changes in its environment by taking extraordinary actions to maintain acceptable system performance

**3.2.113  
retention period**

minimum amount of time that a key or other cryptographic related information should be retained in an archive

**3.2.114  
robustness**

trait describing a system's ability to tolerate significant changes in its environment and still maintain acceptable system performance in such a way as to not require extraordinary actions to be taken

**3.2.115  
secret key**

cryptographic key that is used with a symmetric key cryptographic algorithm that is uniquely associated with one or more entities and that is not made public

**3.2.116  
secure communications protocol**

communication protocol that provides the appropriate confidentiality, authentication, content integrity and message timing protection

**3.2.117  
security association**

secure transport layer association based on:

- 1) source address
- 2) source port
- 3) destination address
- 4) destination port

**3.2.118  
security domain**

system or subsystem that is under the authority of a single trusted authority

NOTE Security domains may be organized (e.g., hierarchically) to form larger domains.

**3.2.119  
security manager**

application software that supervises various operational security aspects of a multi-device network, usually through interaction with device security management objects (DSMO) in the supervised device(s)

NOTE A network security manager may be a dedicated device that is protected both physically and by construction. (See NIST FIPS 140-2 and the NIST/CSE Cryptographic Module Validation Program, <http://csrc.nist.gov/cryptval/cmvp.htm> )

**3.2.120  
security services**

mechanisms used to provide confidentiality, data integrity, authentication and/or non-repudiation of information

**3.2.121  
separation of duties**

security principle that divides critical functions among different staff members in an attempt to ensure that no one individual has enough information or access privilege to perpetrate damaging fraud

**3.2.122****session**

gateway service access point context for gateway resource and communication resource management and usage

**3.2.123****session key, SK, (also known as data key, DK)**

temporary cryptographic key that is used to authenticate or encipher data

**3.2.124****signature generation**

use of a digital signature algorithm and a private key to generate a digital signature on data

**3.2.125****signature authentication**

use of a digital signature algorithm and a public key to verify a digital signature

**3.2.126****source authentication**

process of corroborating that the source of data is as claimed

**3.2.127****split knowledge**

situation wherein a cryptographic key has been split into multiple key components, individually providing no knowledge of the original key other than possibly its size, which subsequently can be combined in any of a number of predefined groupings of the multiple specific keys to recreate the original key

**3.2.128****static key**

key that is not an ephemeral key, which is intended for use for a relatively long period of time and is typically intended for use in many instances of a cryptographic key establishment scheme

**3.2.129****stream cipher**

cryptographic primitive that uses a symmetric key and an initialization vector and that works with continuous streams of input rather than fixed blocks

**3.2.130****subnet**

sub-network, a subset of a full network, either at data-link or network layer (layers 2 or 3 of the OSI Basic Reference Model)

**3.2.131****superframe**

a collection of timeslots with a common repetition period and possibly other common attributes

**3.2.132****symmetric key**

single cryptographic key that is shared by both originator and recipient, for use with a symmetric key algorithm

**3.2.133****symmetric key (cryptographic) algorithm**

cryptographic algorithm that uses the same (usually secret) key for an operation and its inverse (e.g., encryption and decryption)

**3.2.134****system initialization**

with regard to security, setting up and configuring a system for secure operation

**3.2.135****system manager**

application software that supervises various operational aspects of a multi-device network other than security, usually through interaction with network management objects in the supervised device(s)

NOTE 1 A network manager may supervise an entire multi-device network, or may act as a subordinate to another network manager, thereby supervising only a subset of the entire network.

NOTE 2 A network manager need not be a dedicated device.

**3.2.136****threat**

any circumstance or event with the potential to adversely impact plant operations (including function, image, or reputation), assets or individuals through an information system via unauthorized access, destruction, disclosure, modification of data, message delay, and/or denial of service

**3.2.137****true-random bit generator**

process used to seed a pseudo-random bit generator with bits derived from assessing the state of one or more unpredictable physical processes

NOTE 1 Sources of such bits include avalanche breakdown of a Zener diode, shot noise, thermal noise, radioactive decay, cosmic rays, etc.

NOTE 2 Post-processing of such noise sources is required to whiten their output and to detect failures in the circuit providing the randomness. Only the post-processed bit stream is suitable for seeding a pseudo-random bit generator.

**3.2.138****tunneling**

encapsulation of a first protocol within a second communication protocol to convey PDUs from the first protocol

**3.2.139****type-of-service, ToS**

collective name given to a set of protocol elements and associated quality-of-service attributes that together form a subprotocol (e.g., real-time voice, time-critical data, and non-time-critical data) with distinct functionality

**3.2.140****unauthorized disclosure**

event involving the exposure of information to entities not authorized access to the information

**3.2.141****unicast**

messaging from a source to a single intended recipient; see also multicast and broadcast

**3.2.142****user initialization**

process whereby a user prepares the cryptographic application for use (e.g., installing and configuring software and hardware)

**3.2.143****user registration**

process whereby an entity becomes a member of a security domain

**3.2.144  
zeroisation**

method of erasing electronically stored data, cryptographic keys, and critical stored parameters by altering or deleting the contents of storage in a manner that prevents recovery of the data

**3.3 Symbols****3.3.1 Symmetric keys****3.3.1.1 General**

All symmetric keys defined by this standard shall be 128-bit keys. The following defines the list of symmetric keys used. The exact description of the cryptographic material can be found in 0.

**3.3.1.2****K\_DL**

current data key for all devices in the local DL subnet

**3.3.1.3****K\_global**

well-known key whose value is static and published, used to provide uniformity of PDU structure and processing when a shared-secret key is inappropriate or unknown

**3.3.1.4****K\_open**

well-known key whose value is static and published, used to provide uniformity of PDU structure and processing when a shared-secret key is inappropriate or unknown

**3.3.1.5****K\_join**

key used to bootstrap a new device securely into the network

**3.3.1.6****K\_join\_wrapped**

wrapped version of the join key, used to recover from a failed security manager

**3.3.1.7****K\_master**

master key used for key distribution and security management of a single device

**3.3.1.8****K\_session\_AB**

current data key for a session between device A and device B, identical with K\_session\_BA

**3.3.2 Asymmetric keys and certificates****3.3.2.1 General**

All asymmetric keys defined by this standard shall have cryptographic bit strength of 128-bits. The following defines the list of asymmetric keys used. The exact description of the cryptographic material can be found in 0.

**3.3.2.2****CA\_root**

public key of a certificate authority who signed a device's public-key certificate. This key is commonly referred to as a root key and is used to assist in verifying the true identity of the device communicating the certificate, as well as some related keying information.

**3.3.2.3****Cert-A**

public-key certificate of device A, used to evidence the true identity of the device, as well as related keying information, during execution of an authenticated public-key key establishment protocol

**3.4 Abbreviated terms and acronyms**

AES	Advanced encryption standard (see FIPS 197)
AL	Application layer
AME	Application layer management entity
APDU	Application layer protocol data unit
ARMO	Alert reporting management object
ARO	Alert receiving object
ASAP	Application layer service access point
ASDU	Application layer service data unit
ASL	Application sub-layer
ASM	Asset management
ATT	Address translation table
BBR	Backbone router
BECN	Backward explicit congestion notification
CBC	Cipher block chaining stream cipher mode (see NIST SP 800-38C)
CCA	Clear channel assessment
CCM	Counter with cipher block chaining message authentication code
CCM*	CCM enhanced
CIP	Common Industrial Protocol
CON	Concatentation object
CoS	Class of service
CoSt	Change of state
C/S	Client/server
CSMA	Carrier sense multiple access
CSMA/CA	Carrier sense multiple access with collision avoidance
dBm	dB (1 mW)
DBP	Device being provisioned
DCS	Distributed control system
DD	Device description

DDE	Data link layer data (sub-)entity
DDL	Device description language
DDSAP	Data link layer data entity service access point
DIS	Dispersion object
DK	(Symmetric) data key, data authentication key, data encryption key
DL	Data link layer
DME	Data link layer management (sub-)entity
DMIC	Data link layer message integrity code
DMAP	Device management application process
DMO	Device management object
DMSAP	Data link layer management service access point
DPDU	Data link layer protocol data unit
DPSO	Device provisioning service object
DSAP	Data link layer service access point
DSDU	Data link layer service data unit
DSMO	Device security management object
DSSS	Direct sequence spread spectrum
DWT	Data link layer protocol data unit wait time
ECB	Electronic code book
ECC	Elliptic curve cryptography
ECMQV	Menezes-Qu-Vanstone algorithms using elliptic curve cryptography
ECPVS	Pintsov-Vanstone algorithms using elliptic curve cryptography
ECQV	Qu-Vanstone algorithms using elliptic curve cryptography
ED	Energy detection
EDDL	Extended device description language
EUI	Extended unique identifier
FDT/DTM	Field device tool / device type manager
FEC	Forward error correction
FECN	Forward explicit congestion notification
FF-H1	Fieldbus Foundation – H1 Protocol
FF-HSE	Fieldbus Foundation – High Speed Ethernet
FHSS	Frequency hopping spread spectrum

FIPS	[US govt.] Federal information processing standard (issued by NIST)
FPAC	Foreign protocol application communication
FPT	Foreign protocol translator
FPT-PAI	Foreign protocol translator – protocol address information
FPT-PCI	Foreign protocol translator – protocol control information
FPT-PDU	Foreign protocol translator – protocol data unit
FSK	Frequency shift keying
GPS	Global positioning system
GSAP	Gateway service access point
HC	Header compression
HCF	HART Communication Foundation
HMAC	Keyed-hash message authentication code
HMI	Human-machine interface
HRCO	Health reports concentrator object
ICI	Interface control information
ICMP	Internet control messaging protocol
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical & Electronics Engineers
IFO	Interface object
INF	Infinity
I/O	Input/output
IPv4	Internet protocol version 4
IPv6	Internet protocol version 6
ISA	International Society of Automation
ISM	Industrial, scientific, medical
IV	Initialization vector
KEK	Key encryption key
LAN	Local area network
LLC	Logical link control sub-layer (in upper DL)
LQI	Link quality indicator
LSB	Least significant bit
MAC	Media access control layer (spanning lower DL and upper PhL)
MAN	Manual

MIC	Message integrity code
MICI	Media access control interface control information
MK	(Symmetric) master key (used for deriving other keys)
MP	Management process
MPCI	Media access control sub-layer protocol control information
MPDU	Media access control sub-layer protocol data unit
MSB	Most significant bit
NaN	Not a number
NDE	Network layer data (sub-)entity
NDSAP	Network layer data service access point
NFC	Near-field communications
NICI	Network interface control information
NIDS	Network intrusion detection system
NIST	[US govt.] National Institute of Standards and Technology
NL	Network layer
NME	Network layer management (sub-)entity
NMSAP	Network layer management service access point
NO	Native object
NPCI	Network layer protocol control information
NPDU	Network layer protocol data unit
NPDU.F128	The 128-bit final destination address in the expanded network header
NPDU.F16	The 16-bit final destination address in the expanded network header
NPDU.O128	The 128-bit originator address in the expanded network header
NPDU.O16	The 16-bit originator address in the expanded network header
NSAP	Network layer service access point
NSDU	Network layer service data unit
OBJ	Generic application layer object
ODVA	Formerly Open Device Vendor Association; now legally known only by its acronym
OOB	Out-of-band
OPC	Open connectivity in industrial automation
OSI	Open systems interconnection
OTA	Over the air

p-SAP-ID	Provider SAP identifier
PAI	Protocol addressing information
PAR	Positive acknowledgement with retransmission
PCI	Protocol control information
PD	Provisioning device
PDU	Protocol data unit
PhD	Physical layer data service
PhICI	Physical layer interface control information
PhL	Physical layer
PhME	Physical layer management entity
PhMSAP	Physical layer management service access point
PhSAP	Physical layer service access point
PIB	Policy information base
PICS	Protocol implementation conformance statement
PKI	Public key infrastructure
PNO	PROFIBUS Nutzerorganisation (PROFIBUS User Organization)
PPDU	Physical layer protocol data unit
PPM	parts per million ( $10E^{-6}$ )
P/S	Publish/subscribe
PSDU	Physical layer service data unit
QoS	Quality of service
RDP	Reliable datagram protocol
RF	Radio frequency
RFC	Request for comments
RFP	Request for proposal
RSSI	Received signal strength indicator
RSQI	Received signal quality indicator
RT	Routing table
RTO	Retry time-out interval
RTT	Round-trip time
RTTV	Round-trip time variation
SAP	Service access point
SCADA	Supervisory control and data acquisition

SDU	Service data unit
(U)SIM	(Universal) Subscriber identity module
SINR	Signal to interference plus noise ratio
SK	(Symmetric) Session key used for authentication and encryption
SL	Session layer
SMAP	System manager application process
SME	Session layer management entity
SM	System manager
SPDU	Session layer protocol data unit
SRTT	Smoothed round-trip time
S/S	Source/sink
SSAP	Session layer service access point
SSDU	Session layer service data unit
SSM	System security management
TAI	International atomic time; <i>temps atomique international</i>
TCP	Transmission control protocol
TDMA	Time division multiple access
TDE	Transport layer data (sub-)entity
TDSAP	Transport layer data service access point
TFRC	TCP-friendly rate control, IETF RFC 3448
TICI	Transport interface control information
TL	Transport layer
TME	Transport layer management (sub-)entity
TMSAP	Transport layer management service access point
TL-PDU	Transport layer protocol data unit
TMIC	Transport layer message integrity code
Tos	Type of service
TPCI	Transport layer protocol control information
TPDU	Transport layer protocol data unit
TSAP	Transport layer service access point
TSDU	Transport layer service data unit
TUN	Tunnel object
TUN-DATA	Tunnel data

u-SAP-ID	User SAP identifier
UAL	Upper application layer
UAP	User application process
UAPMO	User application process management object
UDO	Upload/download object
UDP	User datagram protocol
UFO	Unified field object
WISN	Wireless industrial sensor network

### 3.5 IEC service table conventions

Portions of this standard use the descriptive conventions given in ISO/IEC 10731.

Service primitives, used to represent service user/service provider interactions (see ISO/IEC 10731), convey parameters that indicate information available in the user/provider interaction.

This standard uses a tabular format to describe the component parameters of the NS (N layer SAP or entity SAP) primitives. The parameters that apply to each group of NS primitives are set out in tables. Each table consists of up to six columns, containing the name of the service parameter, and a column each for those primitives and parameter-transfer directions used by the NS:

- The request primitive's input parameters;
- The indication primitive's output parameters;
- The response primitive's input parameters; and
- The confirm primitive's output parameters.

NOTE The request, indication, response, and confirm primitives are also known as requestor.submit, acceptor.deliver, acceptor.submit, and requestor.deliver primitives, respectively (see ISO/IEC 10731).

One parameter (or part of it) is listed in each row of each table. Under the appropriate service primitive columns, a code is used to specify the type of usage of the parameter on the primitive and parameter direction specified in the column:

- M — parameter is mandatory for the primitive.
- S — selection from defined set of two or more parameters.
- U — parameter is a user option and may or may not be provided depending on the dynamic usage of the NS-user. When not provided, a default value for the parameter is assumed.
- C — parameter is conditional upon other parameters or upon the environment of the NS-user.
- (blank) — parameter is never present.

Some entries are further qualified by items in brackets. These may be:

- a parameter-specific constraint
- (=) indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table.
- an indication that some note applies to the entry
- (n) indicates that the following note n contains additional information pertaining to the parameter and its use.

A summary table is provided to define the parameter usage within the primitive. Each cell defines whether each parameter is mandatory, optional, prohibited, or conditional.

The ordering of the parameters is also implicitly defined from top to bottom.

Complex parameters may also be shown. For example, a structure may be mandatory, but the elements of the structure may be optional. The structure elements are indented by a single space for each level of hierarchy.

Input parameters for services are specified for request and response service primitives. Output parameters for services are specified for indication and confirmation service primitives.

The following abbreviations are used in the service tables:

- .req or Request service request.
- .ind or Indication service indication.
- .rsp or Response service response.
- .cnf or Confirmation service confirmation.

NOTE Intra-device handling of inter-layer error situations, such as situations wherein a lower layer queue may be full, or a lower layer timeout, is a local matter and hence is not addressed by the standard.

## 4 Overview

### 4.1 General

This standard uses the OSI layer description methodology (see Annex C) to define protocol suite specifications, in addition to specifications for the functions of security, management, gateway, and provisioning for an industrial wireless network. The protocol layers supported are the physical layer (PhL), data link layer (DL), network layer (NL), transport layer (TL), and the application layer (AL).

NOTE Although this standard uses the concept of protocol layers, compliance to this standard does not mandate the implementation of these layers.

The wireless network defined by this standard consists of wireless devices serving usage classes 1 through 5 (usage classes are described in Annex C) for non-critical applications of fixed, portable, and moving devices.

References to a network compliant to this standard will hereafter be referred to as a wireless industrial sensor network (WISN).

### 4.2 Interoperability

To ensure vendor-to-vendor interoperability among devices compliant with this standard, all mandatory protocol behavior is described in the body of this standard and summarized in Annex A, along with the minimum capabilities expected from these devices as defined in Annex B. Devices complying with these behaviors and capabilities will be able to form functional wireless networks and interoperate with each other.

### 4.3 Quality of service

To support multiple applications within a network along with diverse needs this standard supports multiple levels of quality of service (QoS). QoS describes parameters such as latency, throughput, and reliability. A device's application(s) requests the level of QoS needed. If the necessary resources can be made available for the requesting application, the system manager will allocate those resources in response to the requested QoS. See Clause 6 for additional information.

### 4.4 Worldwide applicability

This standard is intended to conform to established regulations in all major world regions (e.g., North America, South America, Asia, Europe, Africa, and Australia); however, it may be usable in every regulatory environment.

### 4.5 Network architecture

#### 4.5.1 Interfaces

##### 4.5.1.1 Defined interfaces

This standard defines service access points (SAPs) throughout the protocol layers to allow revisions of any one protocol layer to occur essentially independent of other layers. If, for example, a new PhL is defined that does not require corresponding DL changes, then it can be added to the specification with minimal (if any) impact to the other parts of the standard.

##### 4.5.1.2 Non-defined interfaces

The following interfaces are not defined by this standard:

- System manager to security manager: The security manager and the system manager form two core roles in the network that are closely related. Since the security manager and the system manager are so dependent upon each other and that the security manager can only communicate directly with the security manager; it is believed that most, if not all, implementations would put these two roles into one device or entity. Given that communications will most often be contained within one device or entity, it is not necessary to standardize this interface. This interface may be defined in a future release.
- External interfaces: An important attribute of this standard is that it is designed to allow the wireless network to leverage or integrate into a plant's communication infrastructure.

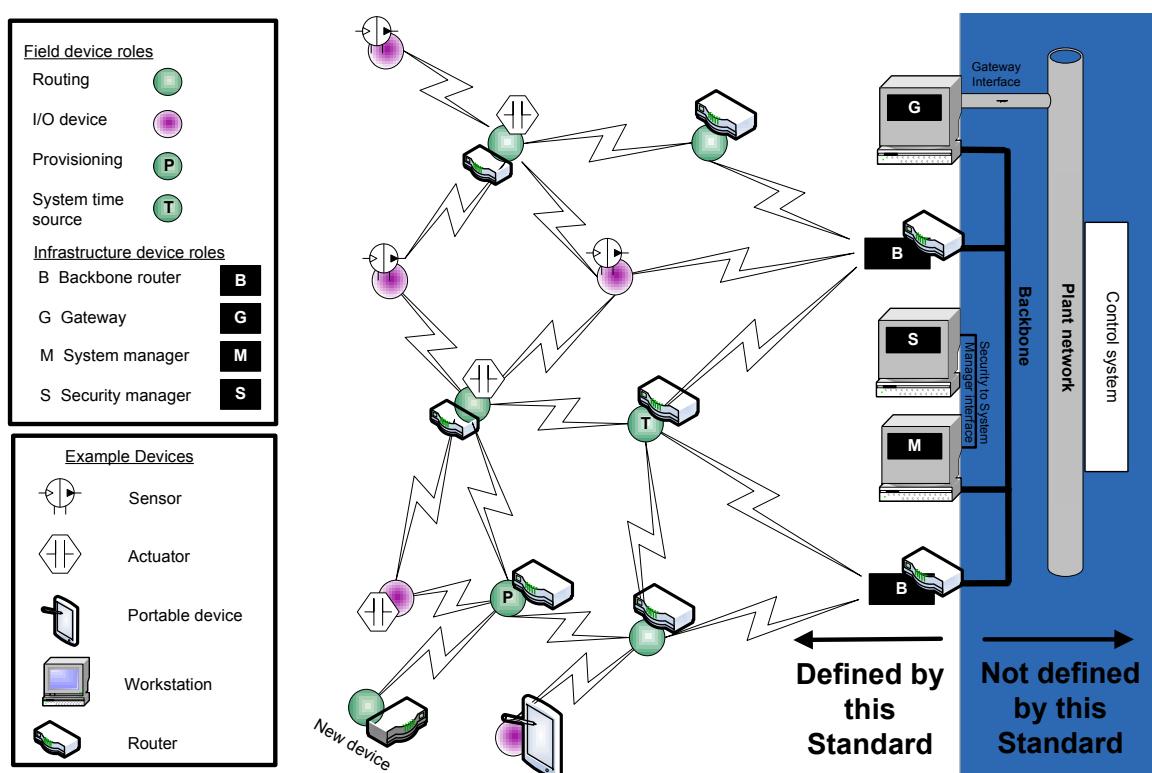
This standard defines specific roles to allow this network to interface to other networks including wired and wireless networks and standard or proprietary networks. However, since external networks cannot be defined in this standard, neither can the interfaces to such networks.

#### 4.5.2 Network description

Figure 1 depicts the communication areas addressed by this standard, as well as those areas (shaded in blue) that are not in scope of this standard. In Figure 1, circular objects represent roles for field devices (sensors, valves, actuators, etc.) and rectangular objects represent roles for infrastructure devices that communicate to other network devices via an interface to the network infrastructure backbone network.

**NOTE** This standard defines roles that devices embody; for further information on these roles, see 5.2.6.

A backbone is a data network (preferably high data rate) that is not defined by this standard. This backbone could be an industrial Ethernet, IEEE 802.11, or any other network within the facility interfacing to the plants network. See Annex E for further information and assumptions about the characteristics of a backbone.



**Figure 1 – Standard-compliant network**

A complete network as defined in this standard includes all components and protocols required to route secure traffic, manage network resources, and integrate with host systems. A complete network consists of one or more field networks that may be connected by an infrastructure device to a plant network.

A field network consists of a collection of field devices that wirelessly communicate using a protocol stack defined by this standard. As shown in Figure 1, some field devices may have routing capabilities, enabling them to forward messages from other devices.

A transit network consists of infrastructure devices on a backbone, such as backbone routers, gateways, system managers, and security managers. Since the backbone physical communication medium and its network protocol stack are outside the scope of this standard, it is not specified and may include tunneling compliant packets over external transport or application layers.

Devices that connect two disparate networks have at least two interfaces. A backbone router has both a field network interface and a backbone interface. A gateway has at least one network interface (either backbone or field network) and a plant network interface.

All addressing, routing, and transport is limited to the scope of the field network. All devices within such a network are identified by a 16-bit local unique ID and are globally identified by a 128-bit address.

#### 4.5.3 Protocol data unit construction

As previously described, this standard uses protocol layers. A protocol layer typically encapsulates its payload, hereafter referred to as a service data unit (SDU), with a header and footer in a single protocol data unit (PDU) as shown in Figure 2. The header and footer are often referred to as overhead, with the amount of overhead depending upon how much information is necessary for the protocol to function properly. Since one of the goals of this standard is to keep PDUs short, minimizing the amount of overhead is a key attribute. A complete description of each header and footer will be found in the appropriate protocol layer clause. A full PDU includes all headers and footers as shown in Figure 3. The amount of data (measured in octets) of an application PDU that can be sent in a single transmission is determined by the difference between the maximum PhL payload and the overhead of all headers and footers.



**Figure 2 – Single protocol data unit**

PhL header	DL header	NL header	TL header	Application PDU	TL footer	NL footer	DL footer
------------	-----------	-----------	-----------	-----------------	-----------	-----------	-----------

**Figure 3 – Full protocol data unit**

#### 4.6 Network characteristics

##### 4.6.1 General

Characteristics of a WISN include:

- Scalable;
- Extensible;
- Support for simple operation;
- License-exempt operation;
- Robustness in the presence of interference and with non-WISNs;
- Determinism or contention-free media access;
- Self-organizing network with support for redundant communications from field device to plant network;

NOTE Redundancy is not defined in this standard, but may be defined in future releases.

- IP-compatible network layer;
- Coexistence with other wireless devices in the industrial workspace;
- Security, including data integrity, encryption, data authenticity, replay protection, and delay protection;
- System management of all communication devices;
- Support for application processes using standard objects; and
- Support for tunneling, i.e., transporting other protocols through the wireless network.

##### 4.6.2 Scalability

The architecture supports wireless systems that span the physical range from a single, small, isolated network, such as might be found in the vicinity of a gas or oil well or a very small

machine shop, to integrated systems of many thousands of devices and multiple networks that can cover a multi-square-km plant. There is no technical limit on the number of devices that can participate in a network that is composed of multiple subnets. A subnet, a group of devices sharing a specific DL configuration, may contain up to 30 000 devices (limitation of subnet addressing space). With multiple subnets, the number of devices in the network can scale linearly.

Although the maximum amount of data that can be transmitted in a single protocol data unit (PDU) is limited by the PhL and amount of overhead, the standard supports fragmentation to transmit a large amount of data. In fragmentation, the data is broken into appropriate network packets at the origination device, transmitted through the network, and then reassembled at the destination device. One use of this mechanism would be for updating firmware on devices.

#### **4.6.3 Extensibility**

The protocols defined by this standard have capabilities such as fields that are reserved for future use and version numbers in headers that allow future revisions to offer additional or enhanced functionality without sacrificing backward compatibility.

#### **4.6.4 Simple operation**

Upon provisioning, as described in Clause 14, a device can automatically join the network. Automatic device joining and network formation enables system configuration with minimal need for personnel with specialized radio frequency (RF) skills or tools.

Additionally, this standard supports the use of fully redundant and self-healing routing techniques to minimize maintenance (see 9.1.6 for further information).

#### **4.6.5 License-exempt operation**

This standard is built upon radios compliant with IEEE Std 802.15.4 (channels 11-26) operating in the 2.4 GHz ISM band, which is available and license-exempt in most countries worldwide.

NOTE Additional information on radio operation may be found in the ISA publication "The automation engineer's guide to wireless technology, part 1: The physics of radio, a tutorial," which is available at [http://www.isa.org/Template.cfm?Section=Find\\_Standards&template=/Ecommerce/ProductDisplay.cfm&ProductID=9238](http://www.isa.org/Template.cfm?Section=Find_Standards&template=/Ecommerce/ProductDisplay.cfm&ProductID=9238).

#### **4.6.6 Robustness in the presence of interference and with non-wireless industrial sensor networks**

This standard supports channel hopping to provide a level of immunity against interference from other RF devices operating in the same band, as well as robustness to mitigate multipath interference effects. In addition, this standard facilitates coexistence with other RF systems with the use of selective channel utilization to detect and avoid using occupied channels within the spectrum. Selective channel utilization can also enhance reliability by avoiding the use of channels with consistently poor performance.

#### **4.6.7 Determinism or contention-free media access**

This standard defines time division multiple access (TDMA) mechanism that allows a device to access the RF medium without having to wait for other devices. By the use of time-synchronized communication using configurable fixed timeslot durations, such as in the range of 10 ms-12 ms, a device is assigned a timeslot and channel unique to that device and the device to which it will communicate. These timeslot durations are configurable on a per-superframe basis. A superframe is a cyclic collection of timeslots. The ability to configure timeslot duration enables:

- Shorter timeslots to take full advantage of optimized implementations
- Longer timeslots to accommodate:
- Extended packet wait times
- Serial acknowledgement from multiple devices (e.g., duocast)
- CSMA at the start of a timeslot (e.g., for prioritized access to shared timeslots)

- Slow hopping periods of extended length

Support is provided for both dedicated time slots for predictable, regular traffic and shared time slots for alarms and bursty traffic. In addition, support is provided for publishing, client/server, alert reporting, and bulk transfer traffic.

#### **4.6.8 Self-organizing networking with support for redundancy**

Fully redundant and self-healing routing techniques, such as mesh routing (see 9.1.6), support end-to-end network reliability in the face of changing RF and environmental conditions. Special characteristics that allow the network to adapt frequencies used (e.g., adaptive hopping) along with mesh routing, can automatically mitigate coexistence issues without user intervention.

#### **4.6.9 Internet protocol-compatible network layer**

This standard's network layer uses header formats that are compatible with the Internet Engineering Task Forces 6LoWPAN standard to facilitate potential use of 6LoWPAN networks as backbone. It should be noted that the use by this standard of headers compatible with 6LoWPAN does not imply that the backbone needs to be based on the Internet Protocol (IP). Furthermore, the use of header formats based on 6LoWPAN and IP does not imply that a network based on this standard is open to internet hacking; in fact, networks based on this standard will typically not even be connected to the Internet.

#### **4.6.10 Coexistence with other radio frequency systems**

##### **4.6.10.1 Coexistence overview**

The system architecture is specifically designed for coexistence with other wireless systems conforming to this standard in addition to other networks operating at 2.4 GHz such as ZigBee™, WirelessHART®, 6LoWPAN<sup>2</sup> protocol networks, IEEE 802.11, Bluetooth™, and RFID systems.

Operating with very short, time-synchronized communications tends to reduce congestion of RF bands and allow neighboring systems to recover quickly from lost packets.

Due to the reduced time of operation on any one channel when channel hopping, the impact on other radio systems is reduced and reliability in the face of interference is increased, since messages can be resent on other, non-interfered channels. Selective channel utilization increases coexistence even further by avoiding those channels that are occupied.

This standard supports the use of clear channel assessment (CCA) to avoid collisions with other non-synchronized systems.

The architecture is designed to support operation in the presence of interference from unintentional radiators, such as microwave ovens, via channel hopping and the use of Automatic Repeat-reQuest (ARQ). ARQ is an error control method for data transmission that uses acknowledgments for successful message reception and delayed retransmission in the case of erroneous reception to achieve reliable data transmission.

For additional information on diversity techniques that minimize coexistence impact, see 9.1.2.

##### **4.6.10.2 Coexistence strategies**

###### **4.6.10.2.1 General**

The following coexistence techniques are not specific to any specific protocols, and should improve coexistence with a wide range of devices sharing the 2.4 GHz band while optimizing first-try success.

---

<sup>2</sup> ZigBee, WirelessHART, and 6LoWPAN are the trademarks of various trade organizations. This information is given for the convenience of users of the standard and does not constitute an endorsement of the trademark holders or any of their products. Compliance to this standard does not require use of the registered trademark. Use of the trademarks requires permission of the trade name holder.

#### **4.6.10.2.2 Leverage infrastructure for high data rate communication links**

Multi-hop networks repeat the same message multiple times. One fundamental capability of this architecture is to get the data to an infrastructure communication link (preferably high data rate with low error rate) as directly as possible, thus typically reducing the use of the standards radio channel to one or two DL messages or DL hops per report.

#### **4.6.10.2.3 Time slotted operation**

Time slotted operation and scheduled transmissions minimize collisions within the subnet, thus avoiding unnecessary use of the channel for retries.

#### **4.6.10.2.4 Radio type selection**

The radio for the PhL was selected since, under many conditions, overlapping and IEEE Std 802.11 radios can transmit simultaneously without loss of data.

#### **4.6.10.2.5 Low duty cycle**

Data transmissions for the focus applications described in the Introduction clause are infrequent. Network overhead is kept to a minimum.

#### **4.6.10.2.6 Staccato transmissions**

Expected transmissions are very short (feature of the selected PhL), i.e., not bursty, enabling co-located IEEE Std 802.11 networks to recover quickly in the event of lost packets.

#### **4.6.10.2.7 Time diversity**

Many of the focused applications have less stringent latency requirements than other users of the spectrum, providing an opportunity to use time diversity for coexistence. Configurable retry periods potentially spanning hundreds of milliseconds enable the system to coexist with other users of the spectrum that may need to use the band for high priority bursts of activity.

#### **4.6.10.2.8 Channel diversity**

The low duty cycle of the radio is spread across up to 16 channels, further reducing the worst-case opportunity for interference to an expected fraction of 1% under many realistic scenarios.

#### **4.6.10.2.9 Spectrum management**

The user may configure superframes within the DL subnet to limit operation to certain radio channels.

#### **4.6.10.2.10 Selective channel utilization**

The DL can avoid problematic channels on a link-by-link basis, such as channels exhibiting IEEE 802.11 cross-interference or persistent multipath fades.

#### **4.6.10.2.11 Collision avoidance**

The DL supports CSMA-CA collision avoidance which, when used, can detect IEEE 802.11 energy and delay its own transmission to reduce interference to IEEE 802.11 networks.

#### **4.6.10.2.12 Changing physical layer payloads for retransmissions**

The probability of an external device mistaking a packet for its own is reduced since retransmissions have different PhL payloads as a result of the DLs inherent security.

### **4.6.11 Robust and flexible security**

All compliant networks have a security manager to manage and authenticate cryptographic keys in transit. Security utilizes security primitives defined by IEEE Std 802.15.4 at the DL and transport layers, providing message authentication, integrity, and optional privacy.

Device authentication is enabled by the use of symmetric keys and unique device IDs, with an option for asymmetric keys.

During normal operation, received data authenticity is verifiable through the use of secret symmetric keys known to both the sender and the receiver.

During provisioning, authenticity of received device credentials from a new device may be verified by a system manager through the optional use of public keys shared openly by the new device, and a corresponding asymmetric secret private key kept inside the new device.

Messages are protected using the default AES-128 block cipher or other locally-mandated cryptographic primitives. Device-to-device communication is secured using symmetric keys.

#### **4.6.12 System management**

This standard includes functions to manage communication resources on each individual device, as well as system resources that impact end-to-end performance. System management provides for policy-based control of the runtime configuration and also monitors and reports on configuration, performance, fault conditions, and operational status. The system management functions take part in activities such as:

- Device joining and leaving the network.
- Reporting of faults that occur in the network.
- Communication configuration.
- Configuration of clock distribution and the setting of system time.
- Device monitoring.
- Performance monitoring and optimization.

System security management works in conjunction with the system management function and optional external security systems to enable secure system operation.

All management functions are accessible remotely via the gateway.

#### **4.6.13 Application process using standard objects**

This standard's application process is represented as a standard object which contains one or more communicating components drawing from a set of standard defined application objects. These objects provide storage and access the data of an application process.

Defining standard objects provide an open representation of the capabilities of a distributed application in a definitive manner, thereby enabling independent implementations to interoperate. Objects are defined to enable not only interaction among field devices but also interoperation with different host systems.

The standard objects and services of this standard may be used to directly map existing legacy field device communications onto standard objects and application sub-layer communication services, thereby providing a means to adapt legacy devices to communicate over the wireless network.

#### **4.6.14 Tunneling**

The native protocols defined by this standard allow devices to encapsulate foreign PDUs and transport these foreign PDUs through the network to the destination device (typically a gateway). This mechanism is referred to as tunneling. Successful application of tunneling will depend upon how well the foreign protocols technical requirements (e.g., timing, latency, etc.) are met by the instantiation of the wireless network. The system clause will more fully describe the techniques to enhance timing and reduce latency.

## 5 Systems

### 5.1 General

In this standard a system is defined to have an application focus and addresses applications and their needs. Networks, on the other hand, have a communication focus and are devoted to the task of device to device communication. For the purposes of this standard, since the focus applications require communication; a network is a part of a system.

This clause describes how the various protocol layers and functions of this standard work together to form a system that achieves the goals of this standard. Specifically, this clause describes the system aspects of devices, networks, protocol suite, data flow, a shared time base, and the applications need for firmware revisions.

### 5.2 Devices

#### 5.2.1 General

A device contains a combination of protocol layers such as PhL, DL, NL, TL, and AL, and may include functions such as the system manager, security manager, gateway, and provisioning.

NOTE Only system behaviors are specified in this clause.

#### 5.2.2 Device interoperability

Device interoperability is the ability of devices from multiple vendors to communicate and maintain the complete network. Device interoperability requires control over the device's various options, configuration settings, and capabilities:

- Options: To allow all devices to interoperate regardless of implemented options (those defined within this standard), devices shall be capable of disabling (i.e., not using) options.
- Configuration settings: The system manager is responsible for configuring WISN devices and roles implemented by WISN devices. The system manager is described in Clause 6.
- Capabilities: There are minimum capabilities to be met for devices based upon their role in the system. Annex B defines the baseline capabilities required for all devices.

#### 5.2.3 Profiles

A profile can be described as a vertical slice through the protocol layers. It defines those options in each protocol layer that are mandatory for that profile. It also defines configurations and parameter ranges for each protocol. The profile concept is used to reduce the risk of device interoperability problems between different manufacturer's products.

A role profile is defined as the baseline capabilities, including any options, settings, and configurations, that are required of a device to perform that role adequately. The roles are defined in 5.2.6.2.

#### 5.2.4 Quality of service

An application within a device is assumed to know the level of service that is necessary for its proper operation. The level of quality of service (QoS) is agreed upon via a contract between the system manager and the requesting device. When the application within a device desires to communicate at a certain QoS level, it sends a request to the system manager notifying it that it wishes to communicate with a specific destination and that it desires a given QoS level. This desired QoS level is indicated by a desired contract and message priority. In addition, a certain level of reliability, periodicity, phase, and deadline for periodic messages, and short-term burst rate, long-term burst rate, and maximum number of outstanding requests for client-server messages, may be indicated. See 6.3.11.2.7 for specific information on QoS.

#### 5.2.5 Device worldwide applicability

The Type A field medium employs a widely used and accepted physical interface and is therefore appropriate for use in many regions of the world. However, regions have special considerations and may be subject to change based upon regulatory needs. Even if a product is designed to meet regulatory requirements, it may not be legally used until it has undergone compliance testing and local permits or certificates have been issued by country-specific

regulatory agencies. This standard does not specify what regulatory certifications or permits a product compliant to this standard has; this is the responsibility of the product manufacturer. A product may have certifications for operation in multiple countries or regions.

It is recommended that the end user verify that the product has the appropriate certifications and permits to operate within the end user's specific application and at the specific application site, particularly if that site is not within the country where the product was purchased or if the product is being applied in an application not intended by the product manufacturer.

## 5.2.6 Device description

### 5.2.6.1 General

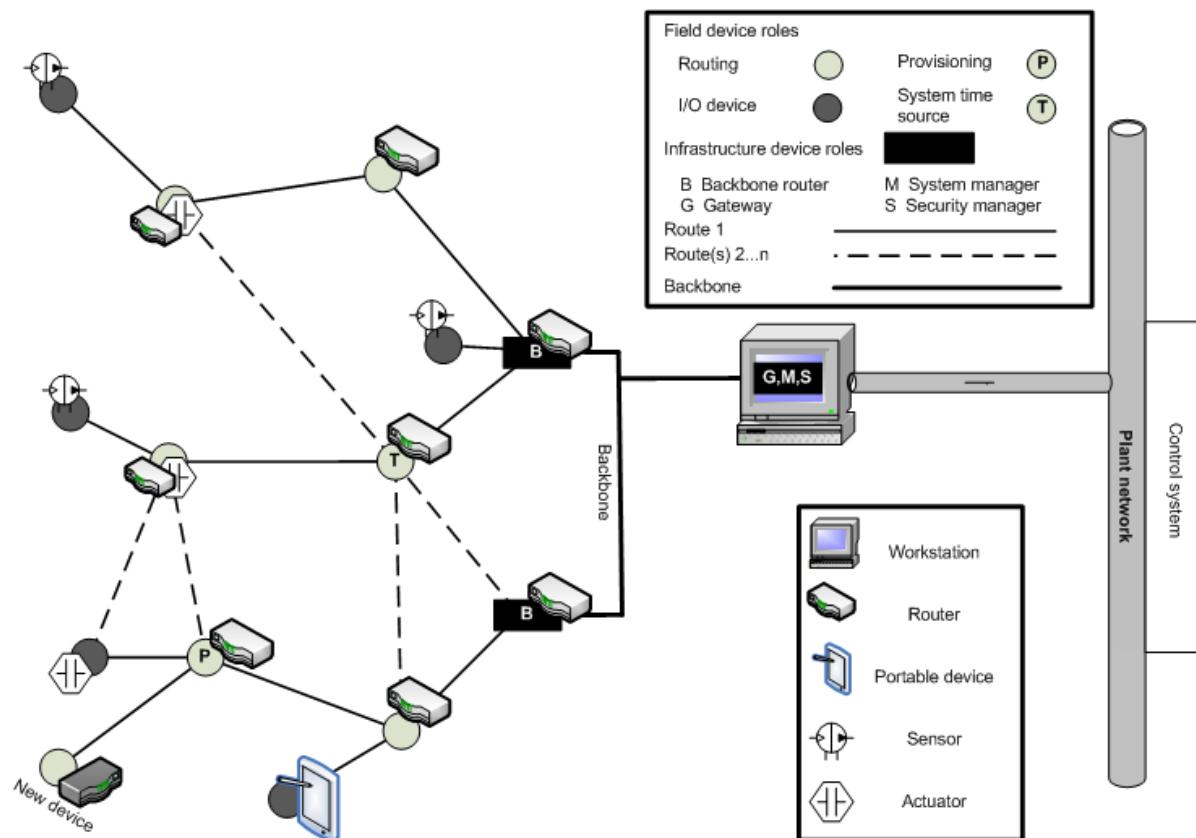
Within this standard, devices are the physical embodiment of the behaviors, configuration settings, and capabilities that are necessary to implement and operate a network. There are many different types of devices depending upon the application, environment, and its function within the network. To fully describe necessary network behavior without defining specific device implementations this standard defines roles, protocol layers, and a field medium that devices may embody.

A role defines a collection of functions and capabilities. This standard defines all the roles necessary for the network to operate properly, including system manager, security manager, gateway, backbone router, system time source, provisioning, router, and I/O device. All devices conforming to this standard shall implement at least one role; however, a device may implement many roles. A device implementing a role shall implement all functions required for that role in 5.3.

The protocol layers describe required behaviors. Not all devices are required to implement all the protocol layers defined in this standard. However, all devices conforming to this standard shall implement the network and transport layers in addition to the DMAP functionality as described in 6.2. Every device shall contain a device management function and a device security management function that cooperate with the system processes to enable secure management of a device's resources and the device's usage of system resources

A field medium is a combination of a PhL and a DL as described in this standard. While not all devices need to implement a field medium; devices implementing the roles of I/O, routing, or backbone routing shall implement at least one field medium defined in this standard.

Figure 4 illustrates the distinction between devices (devices supplied by a manufacturer) and the roles they may assume.



**Figure 4 – Physical devices versus roles**

Figure 4 shows a representative yet complete network compliant with this standard. Within this network are several types of devices, including sensors, actuators, routers, a handheld computer, and a workstation. As shown in Figure 4, each of these devices may assume different roles within the network. For example:

- The workstation has assumed the roles of gateway, system manager, and security manager. These roles are described in 5.2.6.3.5, 5.2.6.3.6, and 5.2.6.3.7, respectively.
- Two devices have assumed the role of backbone routers (described in 5.2.6.3.4), while seven other devices have assumed the role of routers (described in 5.2.6.3.2).
- Three sensors, one actuator and a portable computer have assumed the singular role of an I/O device (5.2.6.3.1).
- The router at the lower left of Figure 4 has assumed a provisioning role, as described in 5.2.6.3.3, and will provision the new device being introduced.
- Two actuator devices have assumed both the router and I/O roles.

NOTE 1 Although Figure 4 shows the use of a backbone network, the functionality of the backbone network is not specified within this standard.

NOTE 2 The physical devices and roles shown in Figure 4 are intended only as examples.

## 5.2.6.2 Field medium

### 5.2.6.2.1 General

This standard defines one specific field medium, Type A. A field medium type defines the protocol for the physical and data link layers. Future revisions of this standard may include multiple field media types.

### 5.2.6.2.2 Type A

The Type A field medium shall consist of the physical layer (Clause 8) and data link layer (Clause 9) as specified by this standard.

Devices implementing the Type A field medium and the data link layer shall implement configuration settings for Radio Silence. The Radio Silence configuration restrains the radio from transmitting during inappropriate times, such as when transmission is unsafe or when regulatory prohibits radio transmissions. The Radio Silence configuration settings are defined in 9.1.15.4.

### **5.2.6.3 Role definitions**

#### **5.2.6.3.1 Input/output**

A device with the I/O role shall provide (source) data to or utilize (consume) data from other devices (and may both provide and utilize data) and shall have at least one user application process (UAP) object. A device with only an I/O role is a device that has the minimum characteristics required to participate in a network compliant with this standard. The I/O role provides no mechanism for the forwarding of messages or routing for any other device. This enables the construction of devices with the least complexity and the potential for low energy consumption, since they need not expend energy routing other devices' messages, nor are they required to accept and provision new devices wishing to join the network.

NOTE A data source supplies data. An actuator would be an example of a consumer of data (i.e. sink), whereas a sensor would supply data (i.e. source).

Devices that implement the I/O role shall implement the Type A field medium.

#### **5.2.6.3.2 Router**

A device with the router role shall have routing capability, shall act as a proxy, and shall have clock propagation capability. These devices can provide range extension for a network and path redundancy and may provide different levels of QoS on a message by message basis. The system manager may disable the routing capabilities of the router role to optimize system performance requirements such as message latency or battery consumption.

Devices that implement the router role shall implement the Type A field medium.

#### **5.2.6.3.3 Provisioning**

A device with the provisioning role (Provisioning device) shall be able to provision a device set to factory defaults and shall implement the device provisioning service object (DPSO; see Clause 14). The provisioning device inserts the required configuration data into a device to allow a device to join a specific network. Devices implementing the PhL shall be capable of being provisioned using the defined physical interface. This capability can be disabled; see Clause 14.

Devices that implement the provisioning role shall implement the Type A field medium.

#### **5.2.6.3.4 Backbone router**

A device with the backbone router role shall have routing capability via the backbone, and shall act as a proxy using the back bone. Backbone routers enable external networks to carry native protocol by encapsulating the PDUs for transport. This will allow a network described by this standard to use other networks, including longer range or higher performance networks.

While the media and protocol suites of backbone networks are not defined in this standard, it is believed that many instantiations of the backbone router will be with Internet Protocol (IP) networks. Many of these backbone networks conform to IPv4 as opposed to the newer IPv6. Clause 10 describes how a WISN PDU coming into a BBR through the BBR's WISN DL interface is converted into a fully compliant IPv6 PDU. If the BBR's backbone interface implements IPv6, then the PDU may simply be routed using standard IPv6. If the BBR's backbone interface implements IPv4, then the BBR shall support the use of IETF RFC 2529 to allow the PDU to be routed across the IPv4 backbone.

Devices implementing the backbone router role shall implement the Type A field medium in addition to the BBR's backbone network interface unless the device also implements the gateway or system manager or security manager roles, in which case the Type A field medium is optional.

### 5.2.6.3.5 Gateway

A device with the gateway role shall implement the high side interface (see Clause 13), shall communicate over the WISN by native access and/or tunneling, and shall have a UAP. The gateway role provides an interface between the WISN and the plant network, or directly to an end application on a plant network. More generally, a gateway marks the transition between communications compliant to this standard and other communications and acts as a protocol translator between the application layer of this standard and other application layers. There can be multiple gateways in a system.

Devices implementing the gateway role shall implement either the router role for access to the Type A field medium or the backbone router role for access to the backbone medium.

### 5.2.6.3.6 System manager

A device implementing the system manager role shall implement the SMAP (6.3.2) and shall set the time source tree.

The system manager is a specialized function that governs the network, devices, and communications. The system manager performs policy-based control of the network runtime configuration, monitors and reports on communication configuration, performance, and operational status, and provides time-related services.

When two devices need to communicate, they do so using a contract. A contract is an agreement between the system manager and a device in the network that involves the allocation of network resources by the system manager to support a particular communication need of this device. This contract is made between the applications in both devices and the system manager. The system manager will assign a contract ID to the contract, and the application within the device will use the contract for communications. An application may only request the creation, modification, or termination to a contract. It is the sole responsibility of the system manager to create, maintain, modify, and terminate the contract.

Devices implementing the system manager role shall implement either the router role for access to the Type A field medium or the backbone router role for access to the backbone medium.

For more information on system management, see Clause 6.

### 5.2.6.3.7 Security manager

The system security management function, or security manager, is a specialized function that works in conjunction with the system manager and optional external security systems to enable secure system operation. The security manager is logically separable and in some use cases will be resident on a separate device and in a separate location. Every system compliant with this standard shall have a security manager. For more information on the security manager, see Clause 7.

NOTE The communication protocol suite between the system manager and the security manager is not defined by this standard.

For more information on security management functionality, see 7.7.

### 5.2.6.3.8 System time source

A device implementing the system time source role shall implement the master time source for the system. A sense of time is an important aspect of this standard; it is used to manage device operation. The system time source provides a sense of time for the entire system. This is described in more detail in 6.3.10.1.

Devices implementing the system time source role shall implement any of the I/O, router, backbone router, system manager, or gateway roles.

## 5.2.7 Device addressing

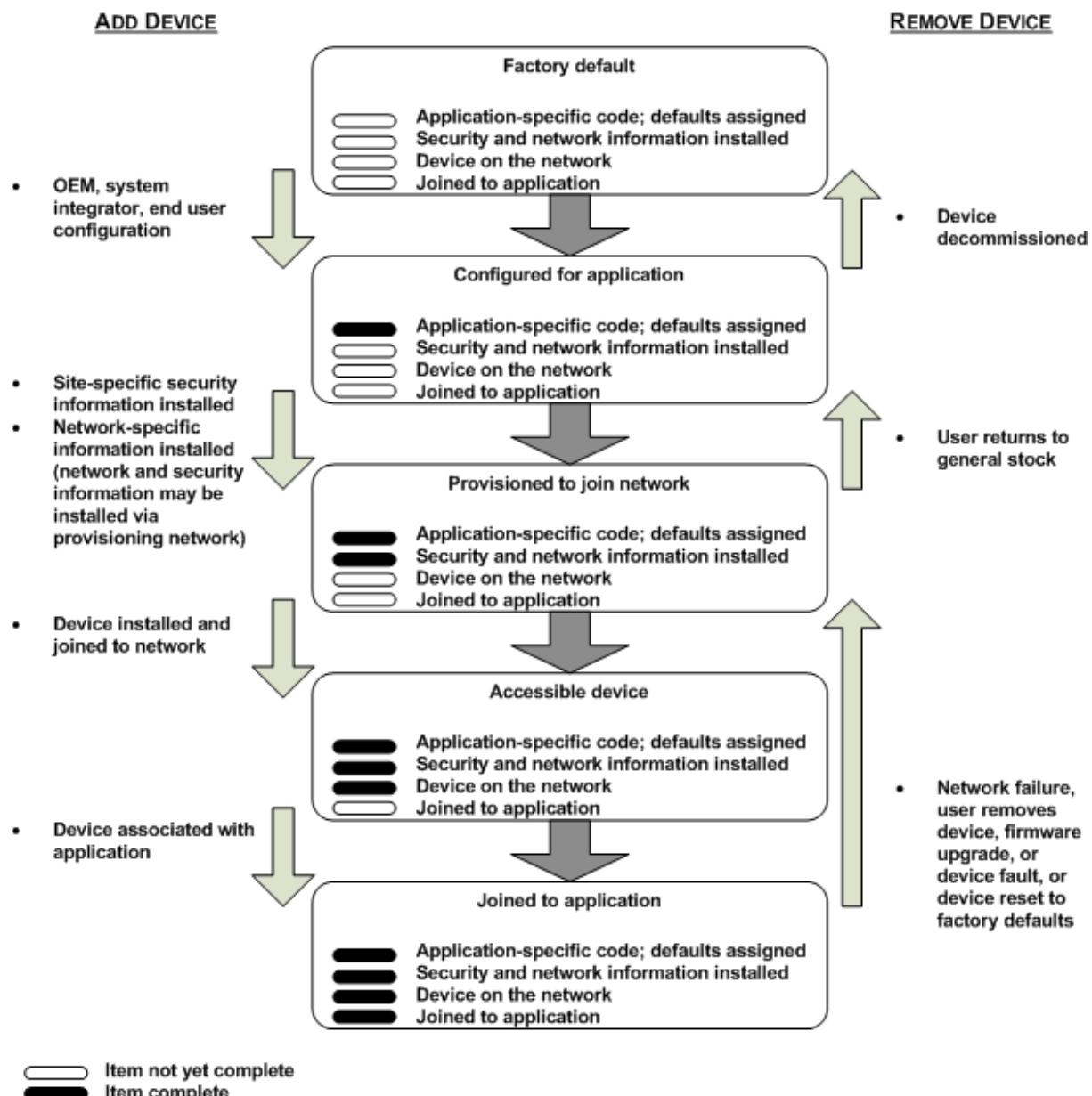
Each device that implements the Type A field medium shall be assigned a 16-bit DL subnet address for local addressing by the system manager. Each device shall have a 64-bit EUI address (EUI-64) that is unique. See Clause 9 for further information.

Each device shall also have a 128-bit network address that is assigned by the system manager as described in 6.3.5. The system manager may choose to assign the 128-bit address as a logical address to maintain application layer linkage in the event of a device replacement. The 128-bit address may be used by the application to reach a particular device within a system after the join process is complete. See Clause 10 for further information.

## 5.2.8 Device phases

### 5.2.8.1 General

A device may go through several phases during its operational lifetime. Within each of these phases are multiple states. A notional representation of the phases of the life of a device is shown in Figure 5. See Figure 166 and Figure 167 for normative detail.



**Figure 5 – Notional representation of device phases**

A device can pass through these phases several times as it is commissioned and used, then decommissioned and re-commissioned for a different application. After joining the network, devices shall be able to report their status so that applications know whether a device is accessible and whether it is joined to an application.

#### **5.2.8.2 Factory default**

A device is considered non-configured if it has not been configured or commissioned with any application- or network-specific information. A non-configured device may come from a manufacturer or may enter a non-configured state as a result of decommissioning.

#### **5.2.8.3 Configured for application**

A device is considered configured for an application when it has received its own application-specific programming and when all appropriate defaults have been applied. A device configured for application may come from a manufacturer or may be supplied by a systems integrator or other value added reseller, already provisioned for the intended application. Over-the-air application program updates can occur, but are handled at the device application layer.

#### **5.2.8.4 Provisioned to join the network**

A device is provisioned to join the network when it has obtained the appropriate security credentials and network-specific information. A device will typically enter this phase when it has been prepared for installation into an application. Typically, a device will not communicate directly with the security manager; instead, the system manager relays requests to the security manager.

#### **5.2.8.5 Accessible device**

A device is considered accessible when it has joined the network and has been authenticated by the system manager. An accessible device can communicate with the system manager.

#### **5.2.8.6 Joined to application**

In this phase, an application object on the device can send or receive information to or from the desired application objects on peer devices. See 7.4 for additional detail on the join process.

NOTE Application objects in any two devices on the network can communicate with one another; refer to Clause 12 for more detail.

#### **5.2.9 Device energy sources**

This standard does not restrict the types of energy sources a device may use. The standard allows for energy efficient device behavior that accommodates device operation for long periods of time (e.g., five to ten years) using suitable batteries.

The types of energy sources may be grouped into five categories:

- Mains
- Limited battery (e.g., button cell)
- Moderate battery (e.g., lead acid)
- Rechargeable battery
- Environmental or energy-scavenging

Devices implementing the roles of I/O or router may be expected to use any category of energy source. The roles of security manager, system manager, gateway, and backbone router are typically performance intensive; therefore devices implementing these roles are recommended to have larger energy sources such as the mains or moderate battery categories.

The energy source status of devices is critical to proper system management. All devices shall provide energy supply information to the system manager. This information may be used in making routing decisions. See Clause 9 for further details.

### **5.3 Networks**

#### **5.3.1 General**

As described earlier, the focus of networks is device-to-device communication. There are numerous aspects of the networks' ability to communicate. These aspects include the atomic

(i.e., minimal or irreducible) network, network topologies, device relationships within a network, protocol suite structure, and the concept of shared time.

### 5.3.2 Minimal network

A minimal network is a network with the minimum amount of devices implementing the minimum number of roles. Although a minimum system could be constructed with just a system manager and a security manager, a more practical minimum system would include the roles of system manager, security manager, provisioning, system time source, and I/O. The system manager and security manager are two separate roles and may reside in the same device or may be split between two physical devices. A single physical device may assume multiple roles. Therefore, a minimal network shall consist of two devices communicating with each other, where one device implements the roles of system manager and security manager; the roles of provisioning, system time source, and I/O are implemented by either of the devices.

A small representative network of four field devices and one infrastructure device is shown in Figure 6. Although such a network is atypical, it represents a small compliant system. In this network, a single physical device has assumed the roles of gateway, system manager and security manager.

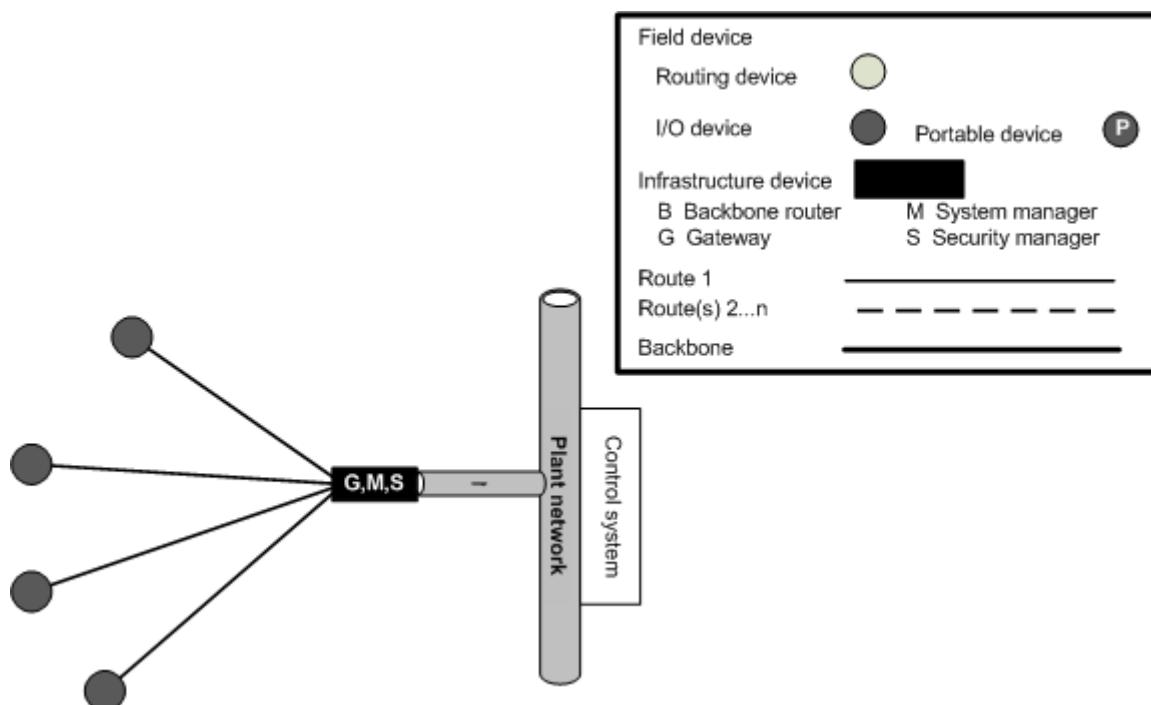
### 5.3.3 Basic network topologies supported

#### 5.3.3.1 General

The figures below provide several informative examples and illustrate the flexibility of the system architecture. The set of examples is not intended to be exhaustive. These examples are presented here only to provide a better understanding of the system elements.

#### 5.3.3.2 Star topology

This standard supports a simple star topology, as shown in Figure 6.



**Figure 6 – Simple star topology**

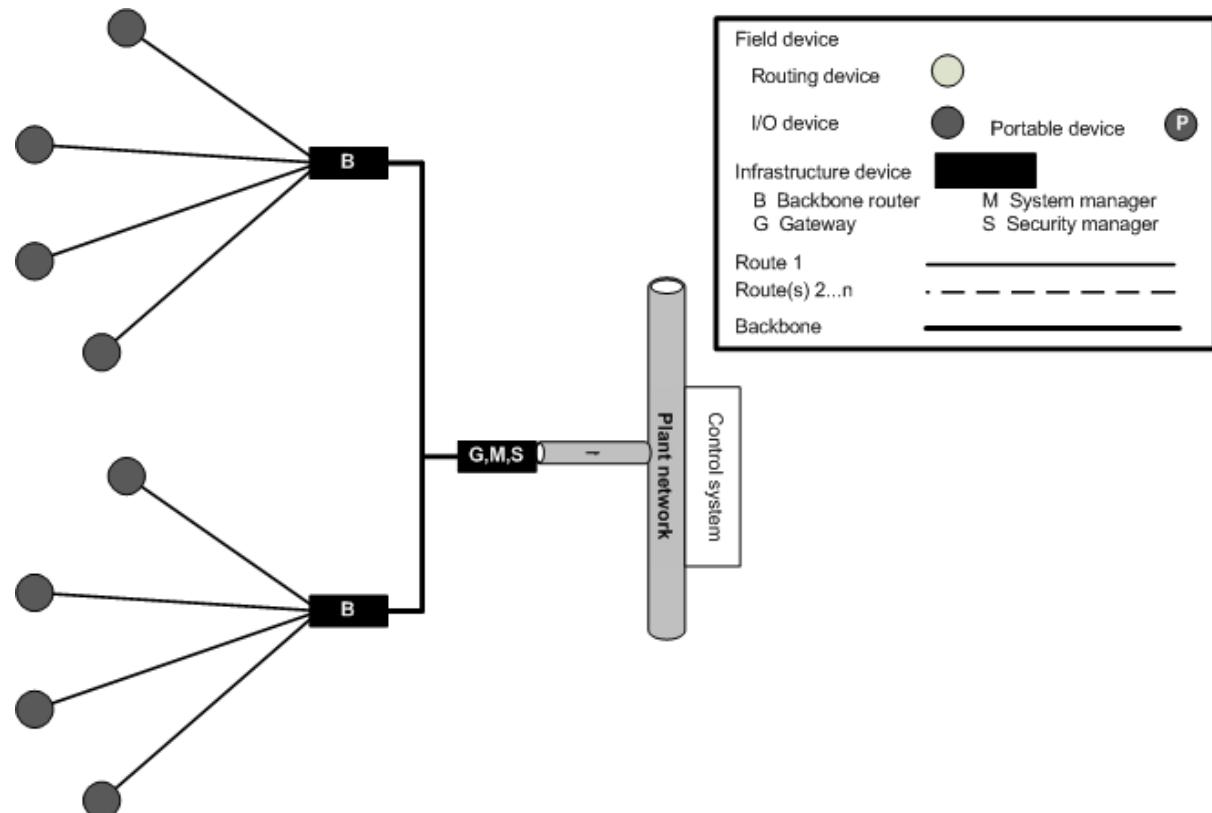
This system configuration can yield the lowest possible latency across the physical layer. It is architecturally very simple, but is limited to the range of a single hop.

In Figure 6 and later figures, the box labeled G,M,S represents a collection of three separate roles combined into one physical device in this simple network:

- A gateway;
- A system manager; and
- A security manager.

### 5.3.3.3 Hub-and-spoke topology

Expanding the network using the backbone routers allow the user to construct a hub-and-spoke network, as shown in Figure 7, wherein devices are clustered around each backbone router, providing access to the high-speed backbone.

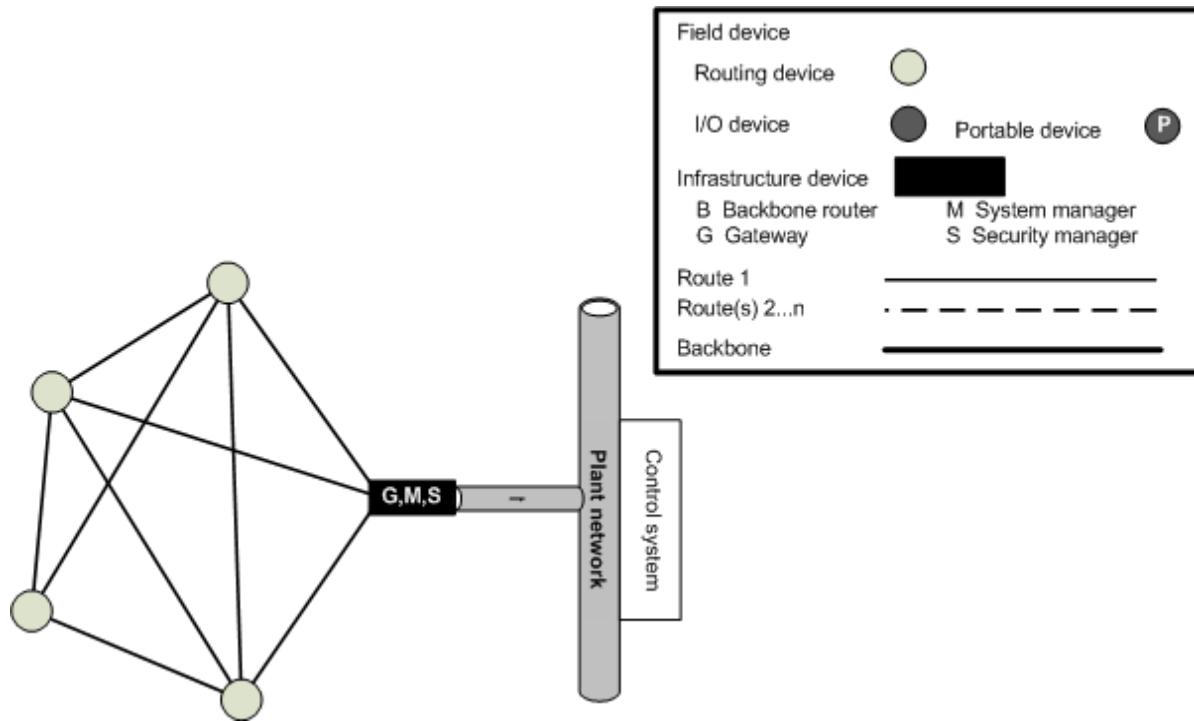


**Figure 7 – Simple hub-and-spoke topology**

In this case, latency is slightly degraded from the simple star topology, but overall throughput can increase, and in larger systems, average latency can decrease because of the multiple data pipes available (one through each backbone router). Although the network can expand further away from the gateway, it is nonetheless limited to single-hop range around a backbone router.

### 5.3.3.4 Mesh topology

This standard supports mesh networking topologies, as shown in Figure 8.



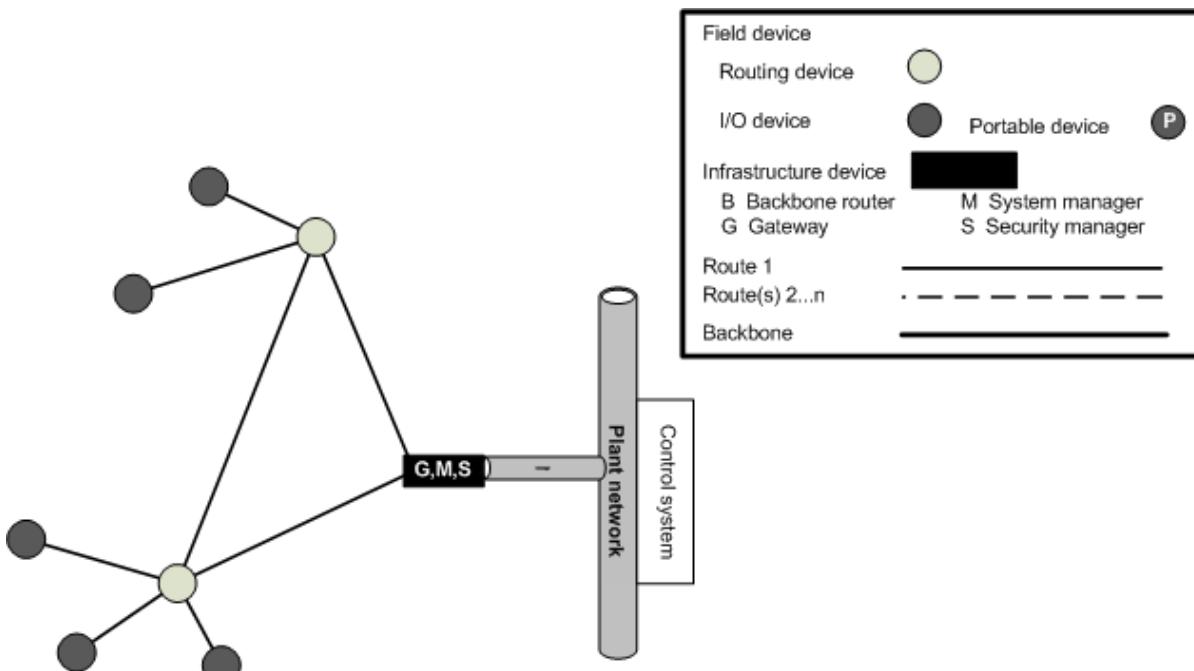
**Figure 8 – Mesh topology**

In some cases, the number of routes a device can support may be limited. Range is extended as multiple hops are supported. Latency is larger, but can be minimized by proper scheduling of transmissions. Throughput is degraded as device resources are used in repeating messages. Reliability may be improved through the use of path diversity.

For more information on mesh topology, see 9.1.14.

### 5.3.3.5 Star-mesh topology

Combining the star topology with the mesh topology is shown in Figure 9.



**Figure 9 – Simple star-mesh topology**

This configuration has the advantage of limiting the number of hops in a network. It does not have the added reliability that full mesh networking can provide.

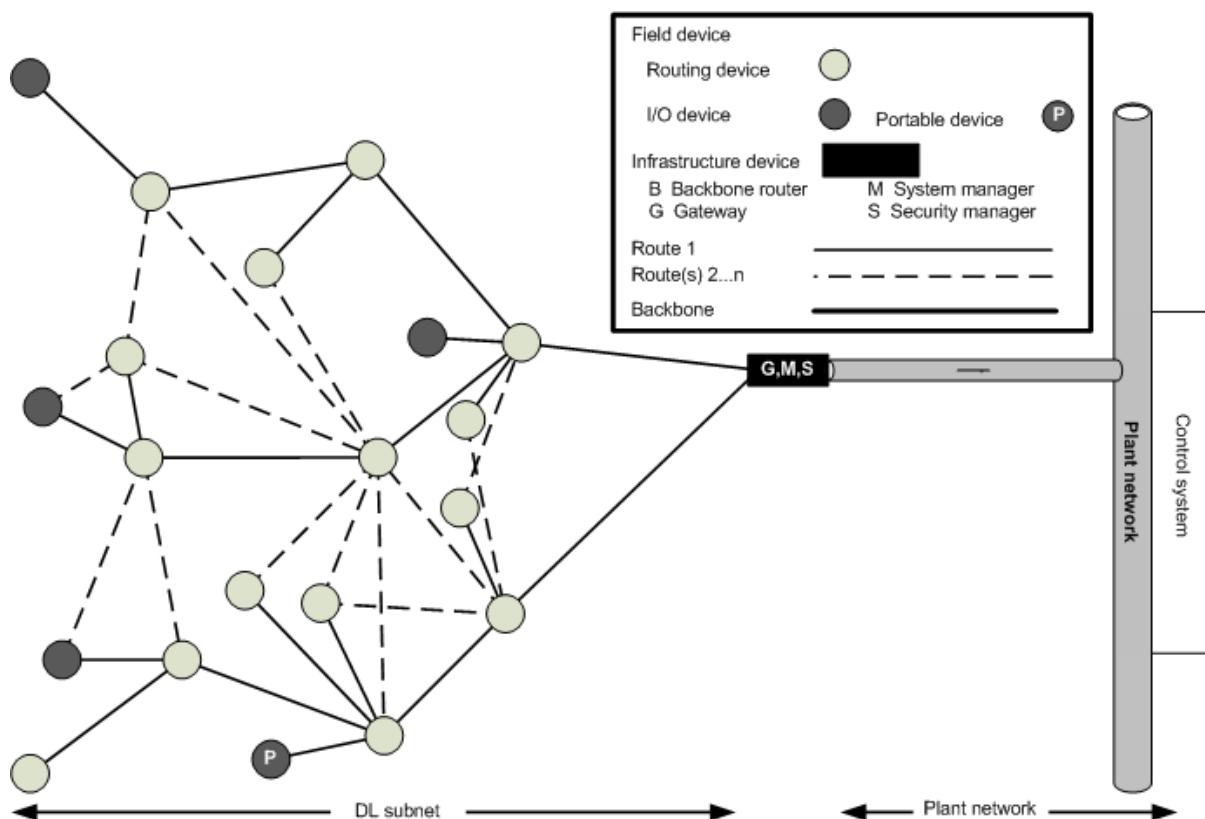
### 5.3.3.6 Combinations of topologies

This standard allows for the combination of any of the previously mentioned topologies, so that a configuration can be constructed that best satisfies the needs of the application. For example, monitoring systems that span large physical areas within a plant may use the star-mesh topology or a combination of hub-and-spoke and star-mesh topologies, whereas certain control applications where latency is critical may benefit from a pure star or hub-and-spoke topology. The flexibility of the system allows for all of these topologies to operate in harmony, in any combination.

### 5.3.4 Network configurations

#### 5.3.4.1 General

The DL subnet in this standard comprises one or more groups of wireless devices, with a shared system manager and (when applicable) a shared backbone. While a DL subnet stops at the backbone router (see 5.5.6), network routing may extend into the backbone and plant network. A complete network includes all related DL subnets, as well as other devices connected via the backbone, such as a gateway, system manager, or security manager. Figure 10 and Figure 11 illustrate the distinction between a DL subnet and a network.



**Figure 10 – Network and DL subnet overlap**

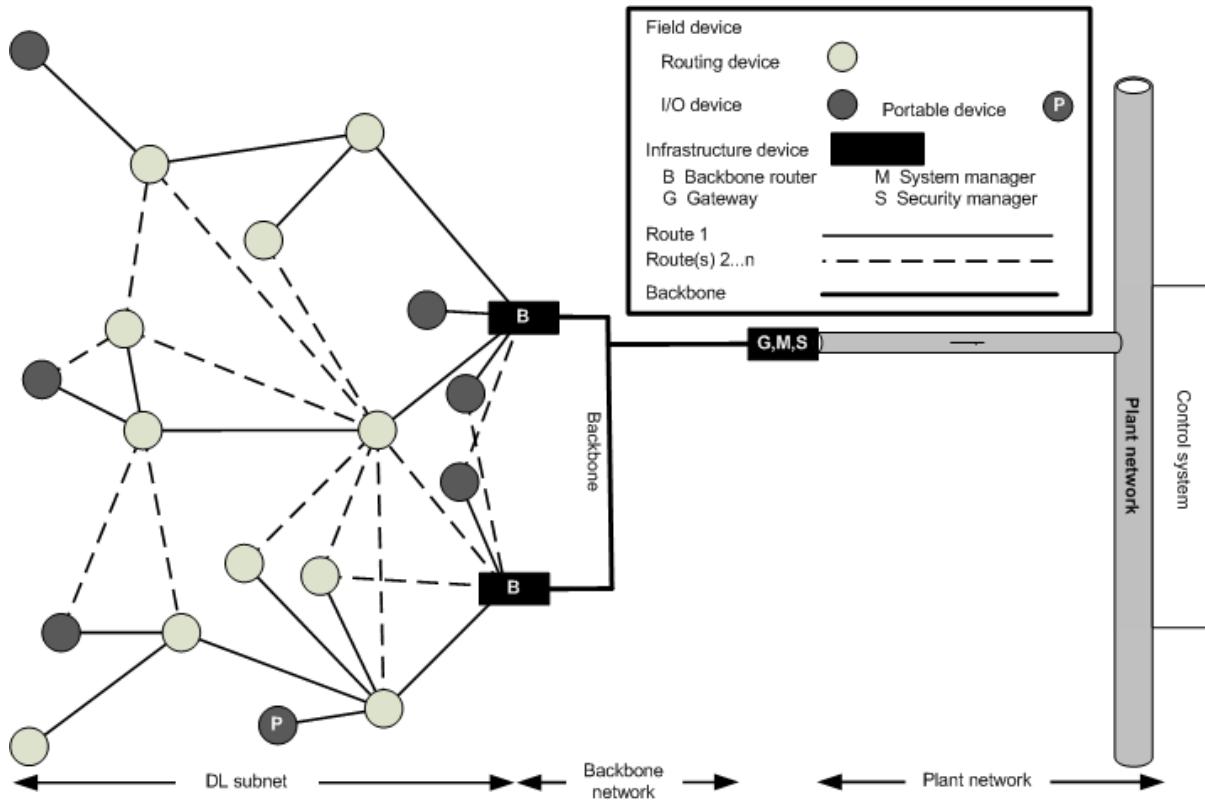
Figure 10 illustrates a simple network comprised of a collection of wireless devices called a DL subnet and additional devices that manage the DL subnet and connect it to other networks. In Figure 10, the network and the DL subnet are the same.

The DL subnet is comprised of both routing and I/O devices. The solid lines between devices designate the first route established between devices, while dotted lines designate the second route, the third route, and so on. Messages may be routed using any one of the known routes.

In Figure 11, the DL subnet includes a collection of field devices up to the backbone routers (boxes labeled B). Backbone routers use connections to a network backbone to reduce the

number of hops that messages would otherwise require, this can improve reliability, reduce latency, and extend the coverage of the network.

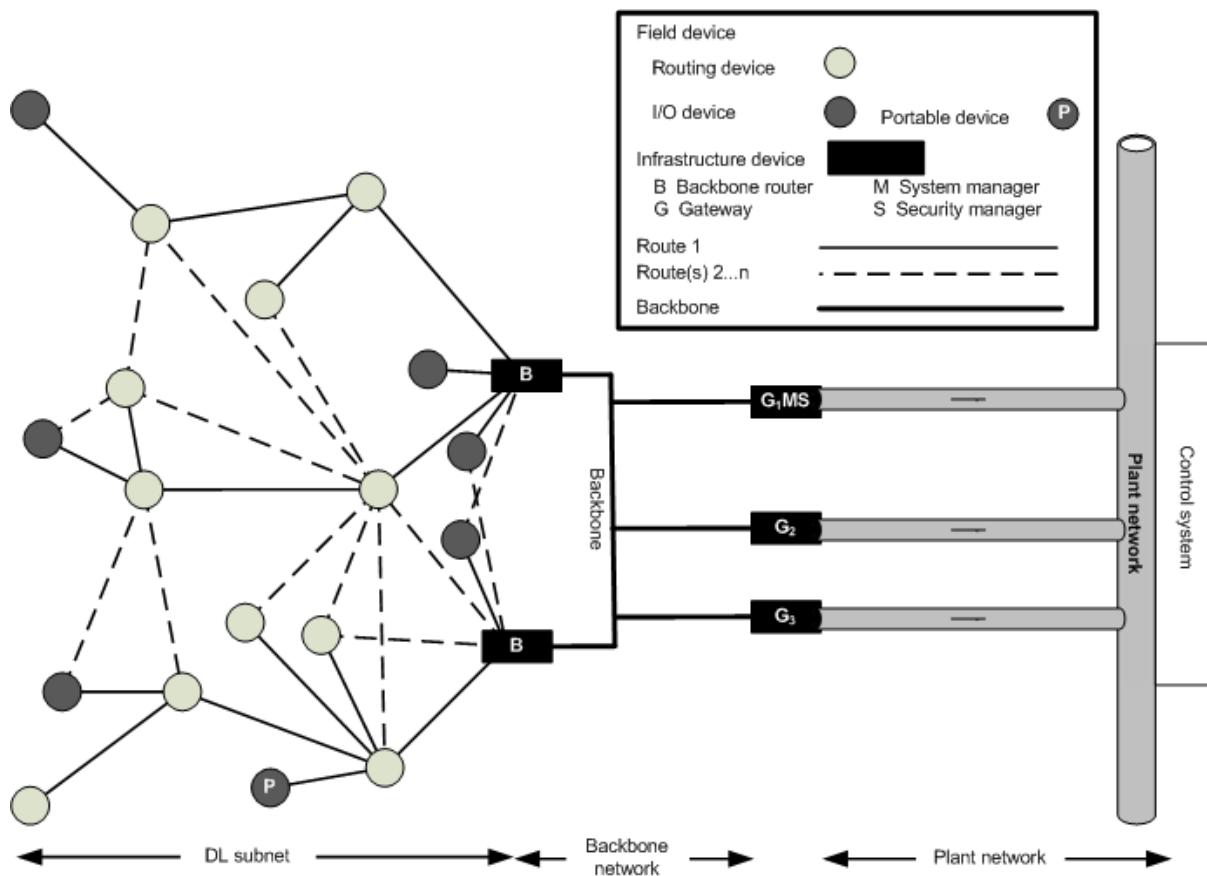
The network in Figure 11 includes the DL subnet, as well as the backbone and a gateway, system manager, and security manager, which are co-located on the backbone.



**Figure 11 – Network and DL subnet differ**

#### 5.3.4.2 Multiple gateways – redundancy and additional functions

Figure 12 illustrates a different physical configuration with three gateway devices. One of the gateway devices also implements the system manager and security manager functions.



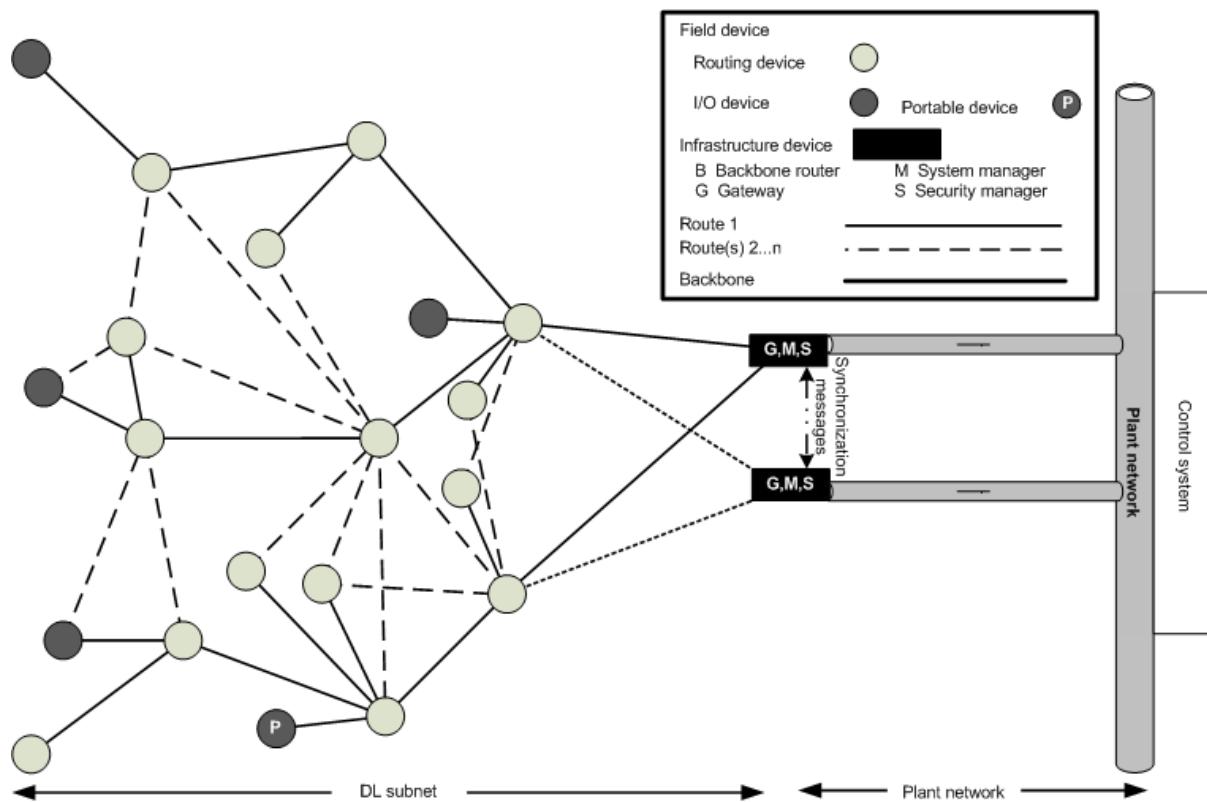
**Figure 12 – Network with multiple gateways**

The gateway devices may be identical (i.e., mirrored, for redundancy) or unique, for example, with each gateway implementing a software application to handle communications between a particular class of device and a control system attached to the plant network.

#### 5.3.4.3 Multiple gateways - designating a gateway as a backup

NOTE This standard does not define the functionality of a backup gateway nor the mechanisms for synchronization of backup gateways.

Figure 13 is similar to Figure 10, but with a second G,M,S device (gateway, system manager, and security manager).

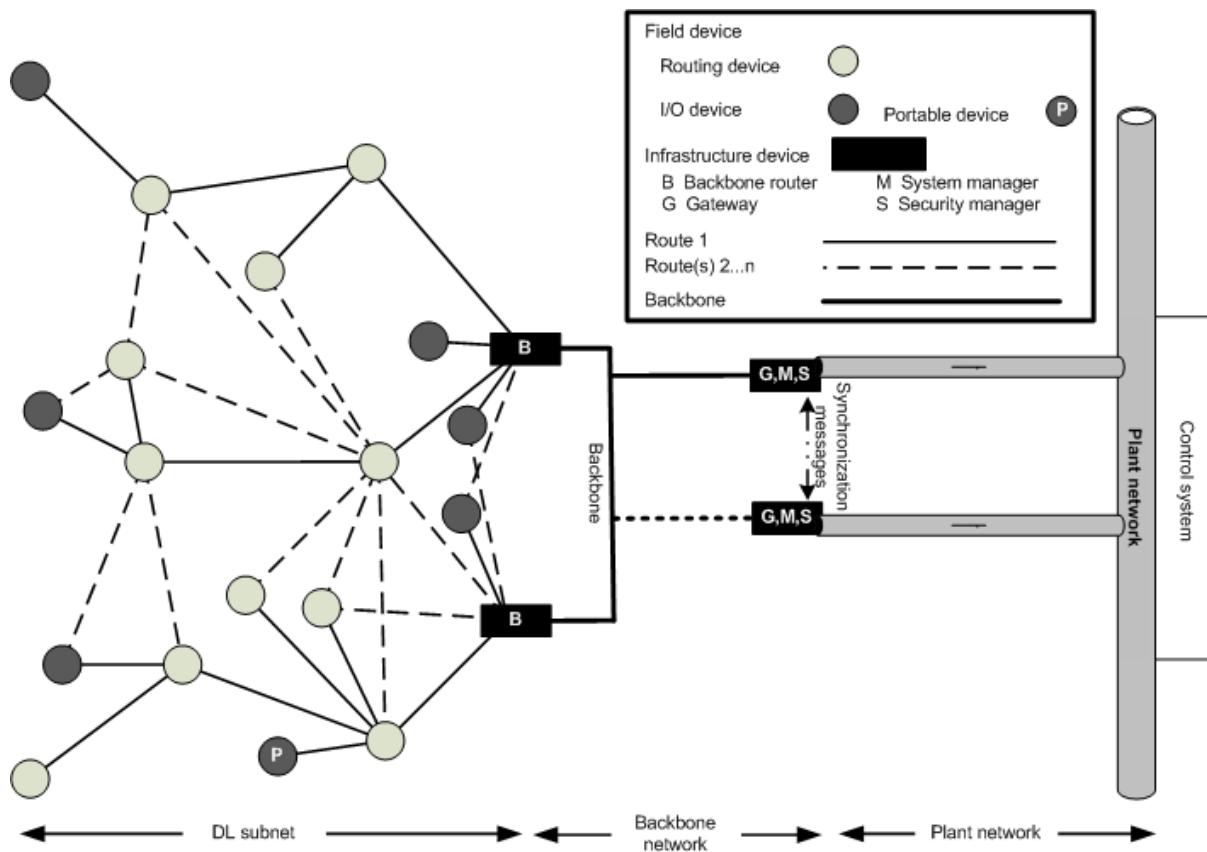


**Figure 13 – Basic network with backup gateway**

The two G,M,S devices offer identical functionality and may coordinate their operation via synchronization messages exchanged through a backchannel mechanism not specified by this standard. A single G,M,S device may be responsible for all gateway, system manager, and security manager functions, with a second G,M,S device acting as an active standby that remains idle until it is needed. Alternatively, the two G,M,S devices may divide the workload between them until one fails.

#### 5.3.4.4 Adding backbone routers

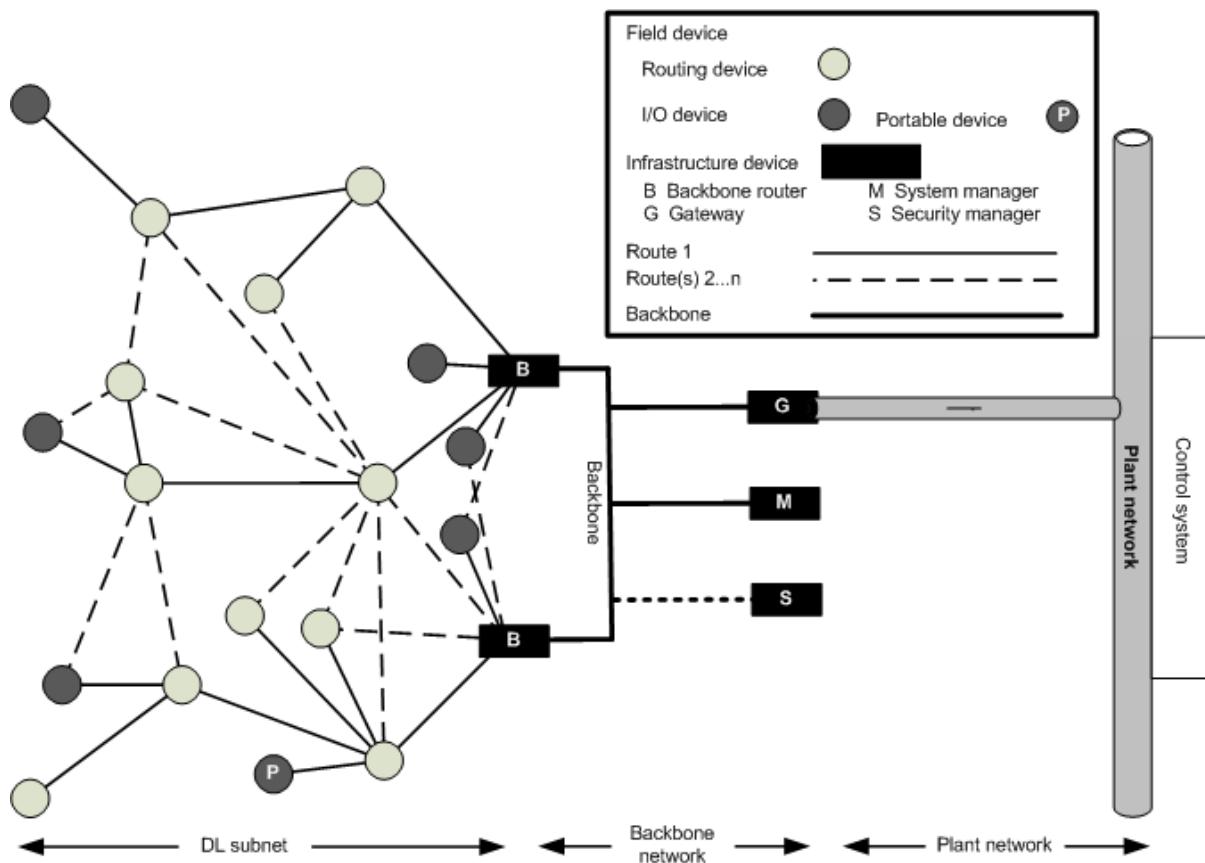
To the basic network shown in Figure 13, Figure 14 adds backbone routers (boxes labeled B), which facilitate expansion of networks compliant with this standard, in terms of both the number of devices and the area the network occupies.



**Figure 14 – Network with backbone**

### 5.3.5 Gateway, system manager, and security manager

As shown in Figure 15, the functional roles fulfilled by the G,M,S device in Figure 10, Figure 11, Figure 12, and Figure 14 may be split into multiple physically separated devices, so that the gateway G, system manager M, and security manager S each operate on a separate device.

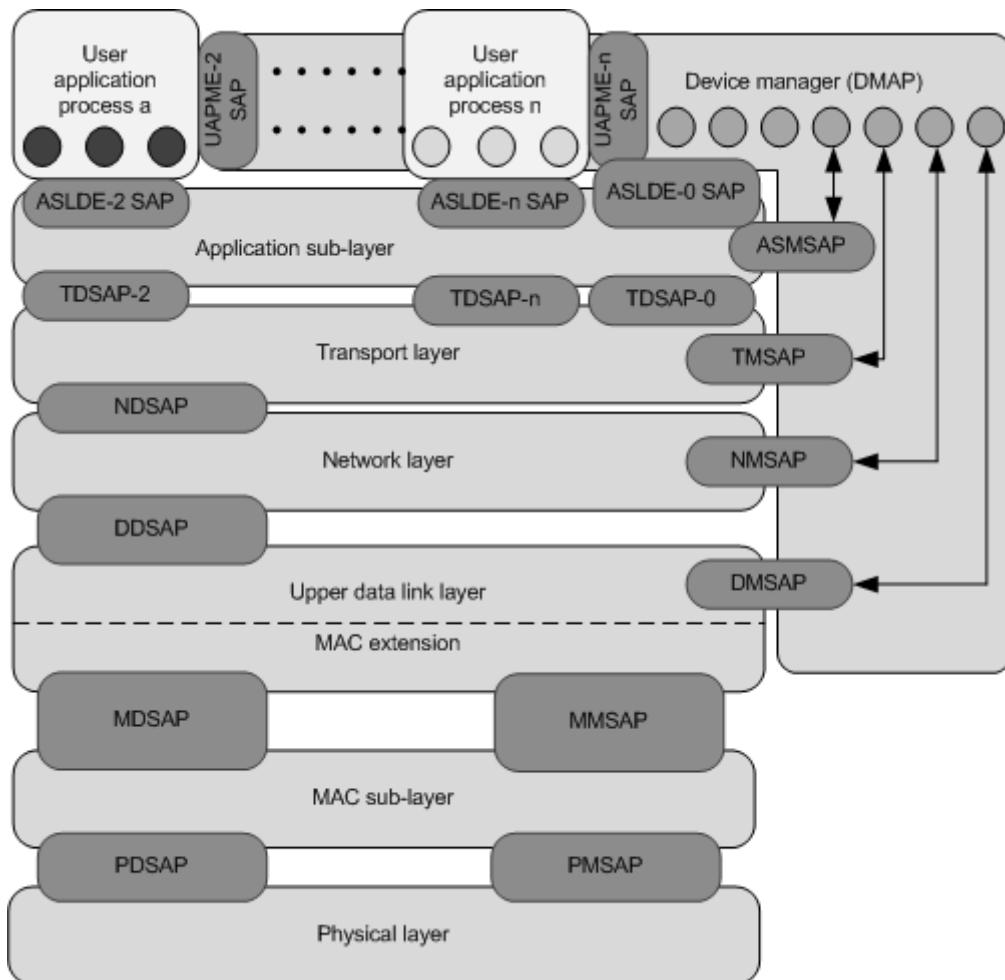


**Figure 15 – Network with backbone – device roles**

The physically separated gateway, system manager, and security manager shown in Figure 15 can be implemented only in networks with a network backbone.

#### 5.4 Protocol suite structure

The protocol layers for a device conforming to this standard are described in terms of the OSI basic reference model, which is adapted as shown in Figure 16. All roles and device types compliant with this standard can be derived from this model by extension or restriction of common elements depicted in Figure 16.



**Figure 16 – Reference model**

As shown in Figure 16, each layer provides a service access point (SAP). The services of a layer are defined as the functions and capabilities of that layer that are exposed through the SAP to the surrounding layers. The services provided by a layer are defined by the data flowing through the SAP and, in some cases, the states that a layer provides and the state transitions that are driven by the interaction across the SAP. The device manager is the entity within each device that performs the management function. Note that the device manager has a dedicated path to several of the lower protocol layers within a device. This is to provide direct control over the operation of these layers, as well as to provide direct access to diagnostics and status information.

All devices compliant to this standard are considered managed devices. All devices shall implement each SAP used by the DMAP for every protocol layer they implement, as shown in Figure 16.

## 5.5 Data flow

### 5.5.1 General

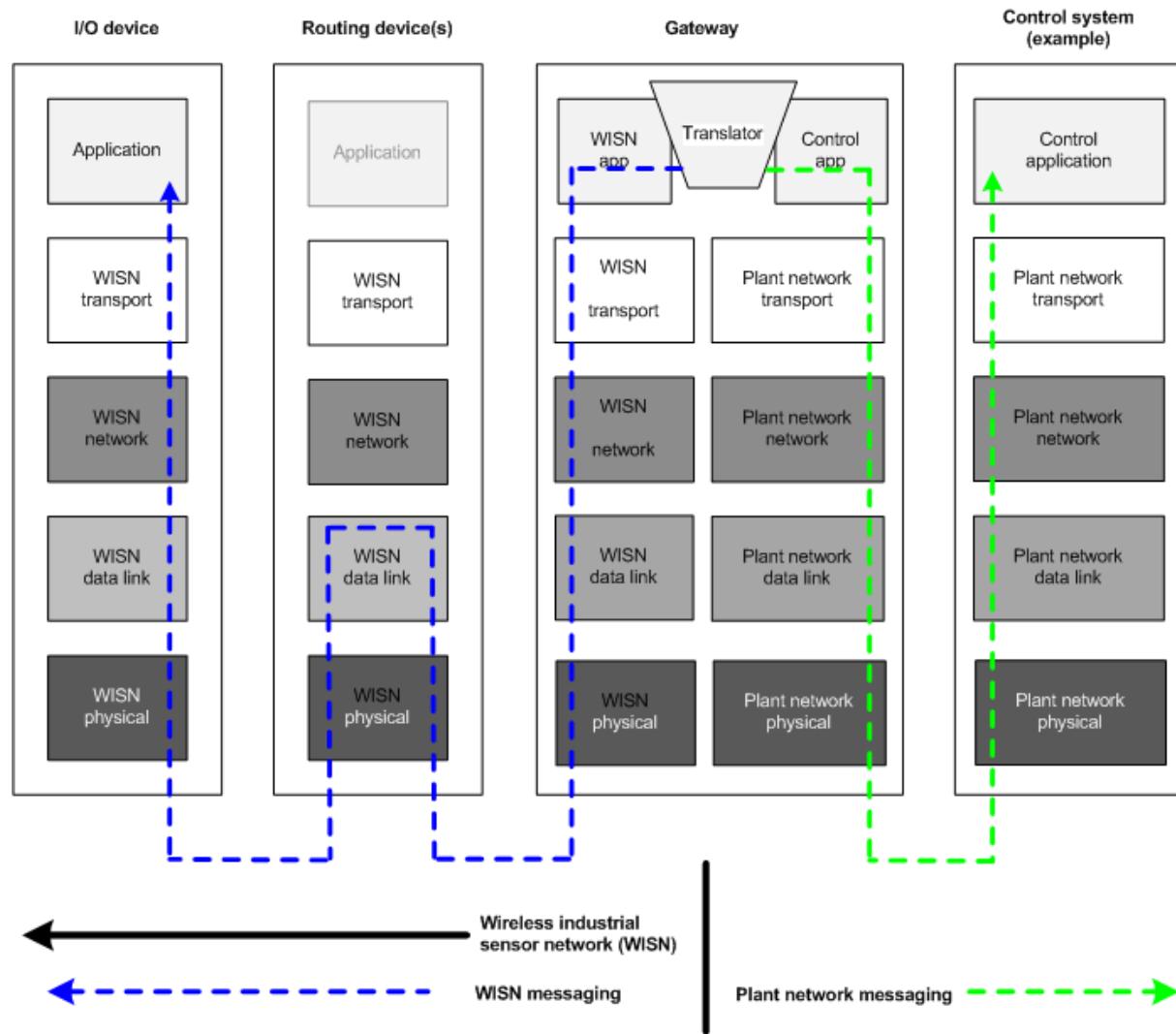
The following descriptions are intended to provide examples of how data may flow through the system. The set of examples is not intended to be exhaustive.

### 5.5.2 Native communications

A device communicates over the network using only ASL defined services as defined in Clause 12; the payloads are classified as either native or non-native. Native payloads are defined in Clause 12; non-native payloads are not defined within this standard.

### 5.5.3 Basic data flow

Figure 17 illustrates the steady-state data flow for a basic network compliant with this standard, such as the one shown in Figure 10.



**Figure 17 – Basic data flow**

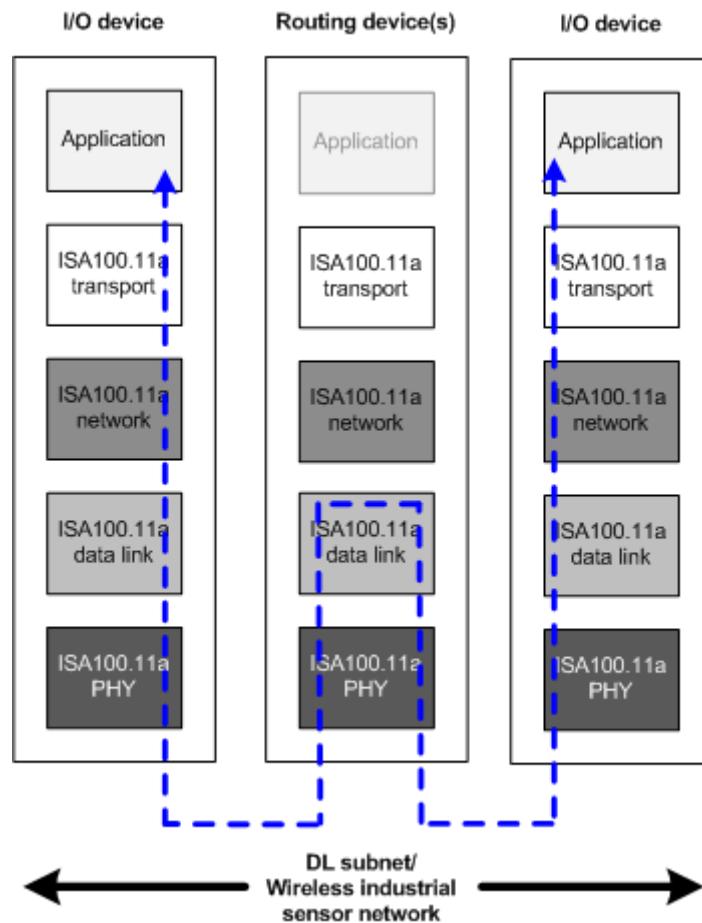
The I/O device is a sensor or actuator device within the DL subnet that contains physical, data link, network, and transport layers as defined by this standard and runs an application that handles the sensor or actuator function.

The router routes messages on behalf of the I/O device. Routing within the DL subnet is performed entirely at the data link layer, not at the network layer; see Clause 9. In a real-world network, there will be one router for each additional hop between the device and the gateway or backbone router.

The gateway translates messages between the DL subnet and the plant network. The application running on the gateway consists of a component that communicates with the application layer of the I/O device, plus a component that communicates with the application layer of the control system, plus any components that facilitate translation between the two, such as a cache.

### 5.5.4 Data flow between I/O devices

Figure 18 illustrates the data flow for communication between I/O devices within a DL subnet. Routing within the DL subnet is performed entirely at the data link layer, not at the network layer.



**Figure 18 – Data flow between I/O devices**

#### 5.5.5 Data flow with legacy I/O device

Figure 19 illustrates a legacy I/O device that is integrated into a DL subnet via a legacy device adapter. An adapter is a subset of the gateway role and is a device that converts the protocol in the legacy device to that of a network compliant with this standard.

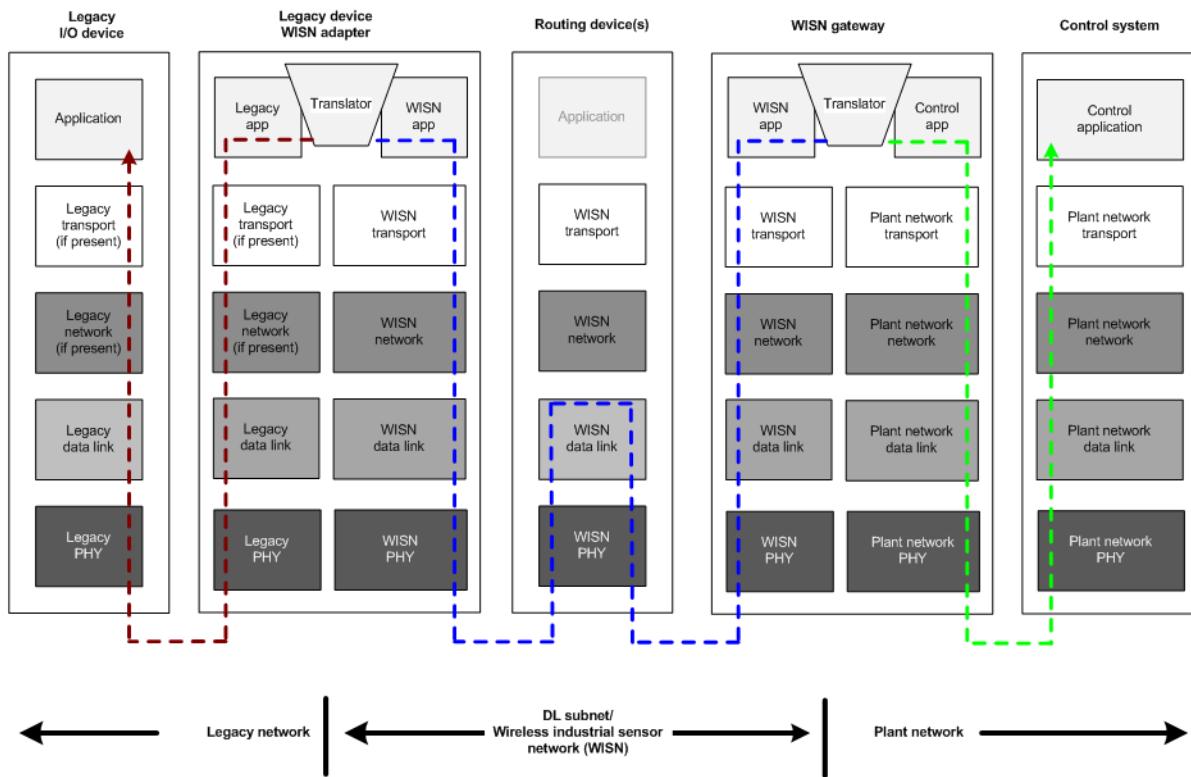


Figure 19 – Data flow with legacy I/O device

### 5.5.6 Data flow with backbone

Figure 20 introduces a backbone router into the data flow.

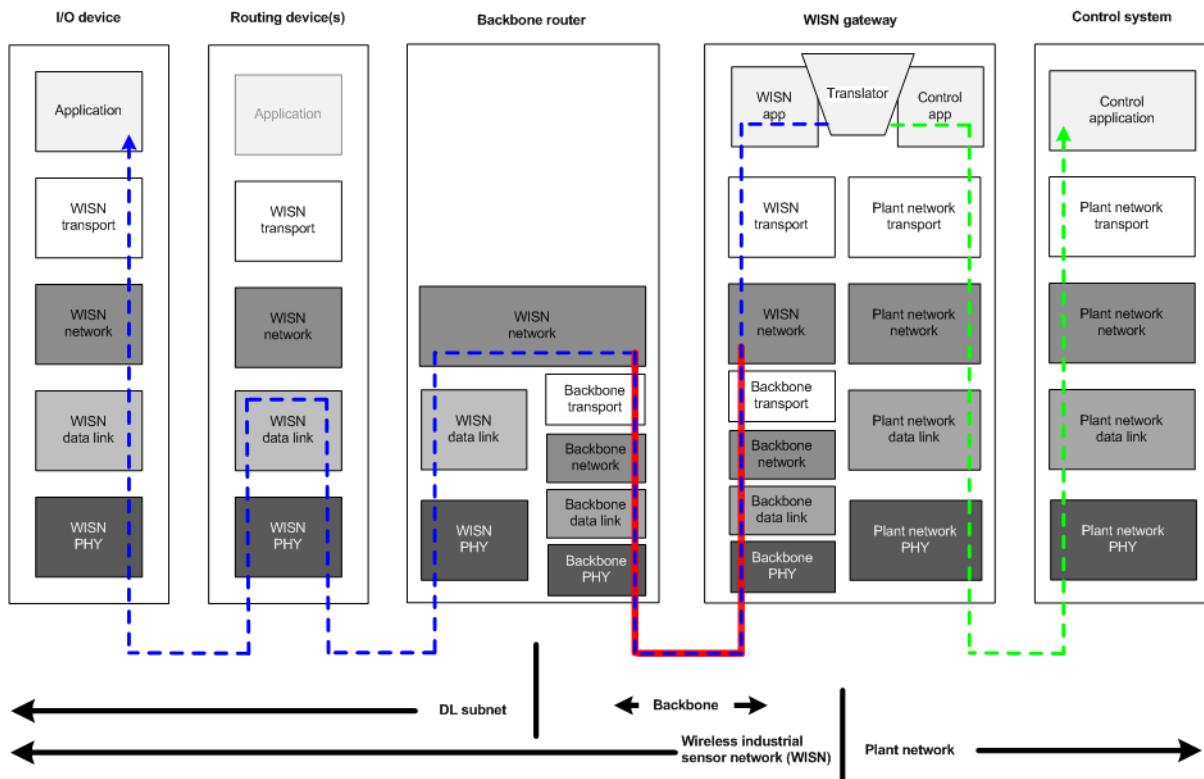


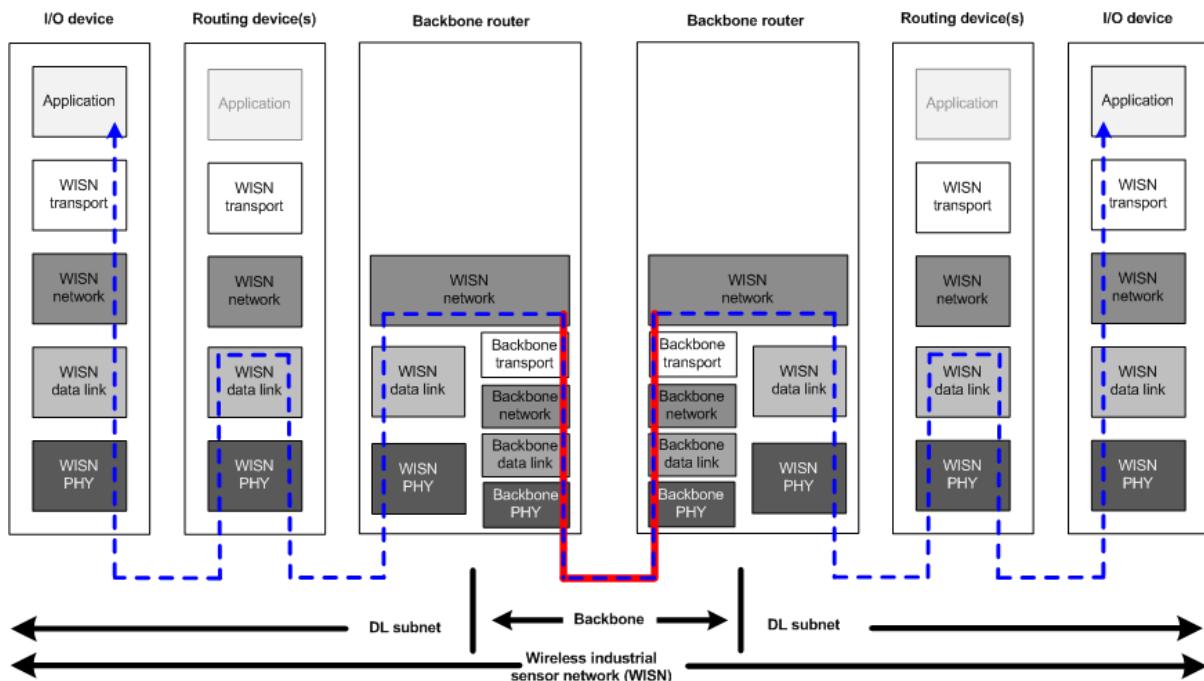
Figure 20 – Data flow with backbone

The backbone router encapsulates network layer messages and relays them through backbone physical, data link, network, and transport layers. The gateway uses the same

backbone layers to recover the network layer messages. While it is not shown in Figure 20, the gateway may include both physical and data link layers as defined by this standard, enabling the gateway to handle messages directly from a DL subnet, in addition to messages relayed through the backbone interface.

### 5.5.7 Data flow between I/O devices via backbone

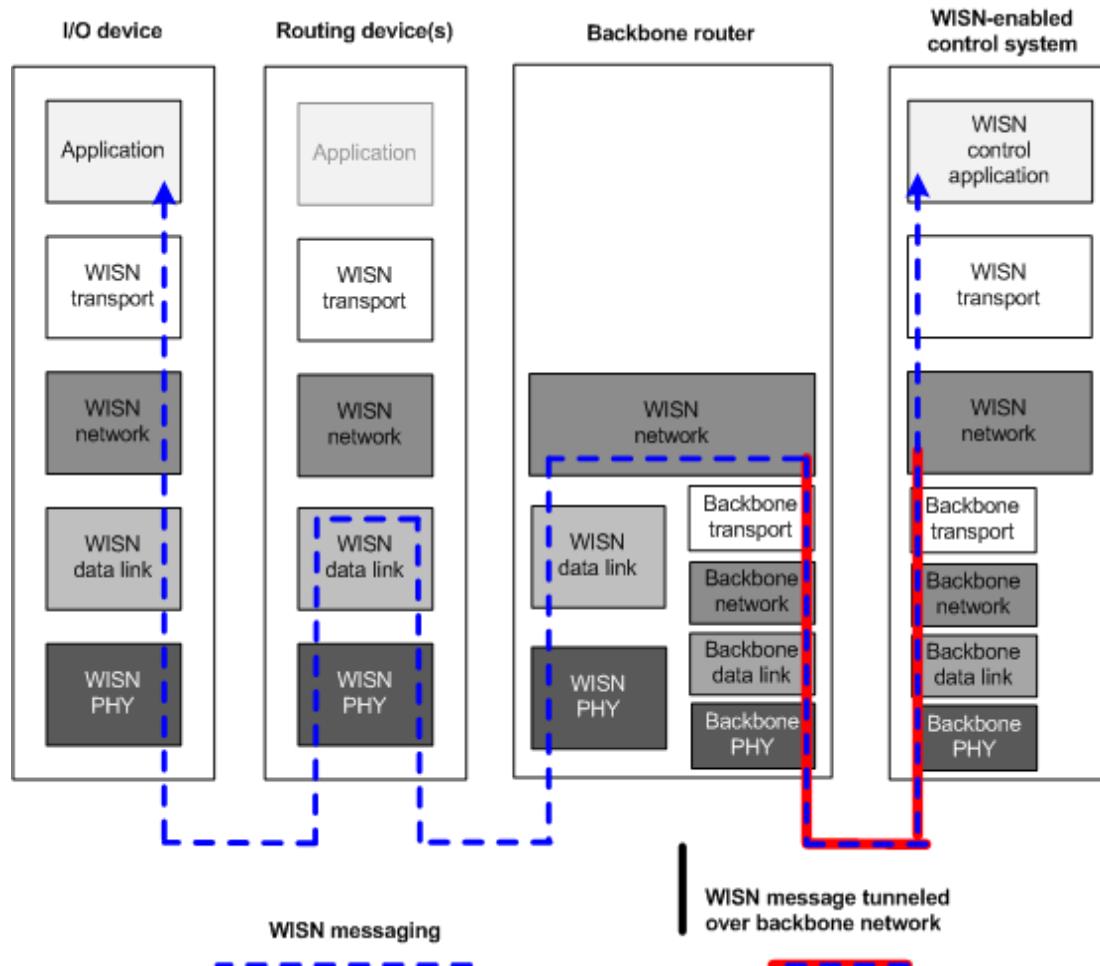
Figure 21 illustrates how a backbone network handles standard-compliant message transfer when an I/O device communicates directly with another I/O device in a different DL subnet.



**Figure 21 – Data flow between I/O devices via backbone**

### 5.5.8 Data flow to a standard-aware control system or device

A standard-aware control system is a control system that understands messaging defined by this standard and does not need a gateway to perform protocol translation. Figure 22 illustrates data flow to a standard-aware control system.



**Figure 22 – Data flow to standard-aware control system**

In general, for a device to be standard-aware, it needs only to support the application interface defined by this standard and to implement the application, transport, and network layers defined by this standard. This makes it possible for two standard-aware devices to communicate via a plant network without utilizing or requiring any sub-layers.

## 5.6 Time reference

### 5.6.1 General

This standard's time is based on International Atomic Time (TAI) as the time reference; see 6.3.10. This standard's time is reported as elapsed seconds since the TAI instant of 00:00 on 1 January 1958 (i.e., 1958/01/01 00:00).

It is not possible or even desirable for every network to track an atomic clock precisely. Rather, every network shall have a sense of time that is:

- Monotonically increasing at a rate that closely matches real time;
- Not to exceed an error of more than 1 s relative to the system time source; and
- Delivered to various layers in field devices in consistent TAI units.

There are communication modes as defined in Clause 9 that require better than 1 s clock accuracy relative to the system time source.

For protocol operation, sequence of event reporting, and other purposes, time typically needs to be divided into increments of less than one second. For example, increments may be represented as:

- Clock ticks: Two octets to mark time in increments of  $2^{-15}$  s (32 768 Hz, or  $\sim 30.52 \mu\text{s}$  per tick).

- Microsecond precision: Three octets to mark time in increments of  $2^{-20}$  s (~0.95 µs per increment).
- Nanosecond precision: Four octets to mark time in increments of  $2^{-30}$  s (~0.93 ns per increment).

Devices needing to convert TAI time to hh:mm:ss format, such as on a user display, may account for a Coordinated Universal Time (UTC) accumulated leap second adjustment. This adjustment is available to field devices from the system manager. If used, the UTC adjustment should be refreshed by a field device at the start of each month.

NOTE 1 Simultaneous UTC update requests by many devices may cause a storm of activity in the DL. This should be considered in the DMAP design and is not covered by the DL specification at this time.

NOTE 2 A list of such adjustments is maintained at <ftp://maia.usno.navy.mil/ser7/tai-utc.dat>.

All devices in a network share the TAI time reference with variable degrees of accuracy. All devices within a network shall be accurate to within 1 s.

The system manager will direct devices on the system to a device implementing the role of system time source. In most cases, this will also be the system manager, but this responsibility can be redirected to any device with a more capable source of time.

The gateway shall be responsible for converting nominal network TAI time to an external time source if one is available.

For more information on the requirements for the time source, see 6.3.10.

### **5.6.2 Time synchronization**

To propagate host time, a gateway may perform periodic synchronization of time in an attached DL subnet time to an external source by requesting time changes through a DLMO.

The system supports time synchronization so that, at the device level, applications may use time to coordinate activities or time-stamp information, this activity could improve energy use and enhance reliability. System time shall be available from at least one device (system time source) on the network. See Clause 9.

### **5.7 Firmware upgrades**

The system shall provide the capability to upgrade the firmware used to implement this standard for each device on the WISN via the wireless network (see 6.3.6). The system shall support a common mechanism to inform all devices to switch to new firmware simultaneously; this mechanism may be used to assure that no devices are left stranded with an incompatible network protocol suite. The security mechanisms built into this standard are used during a firmware upgrade.

NOTE This mechanism may use time as a trigger.

### **5.8 Wireless backbones and other infrastructures**

Devices compliant with this standard are managed devices. All devices compliant with this standard shall implement the device management interfaces at each layer, but they may implement only the functionality of their required functional layers.

The system supports both wired and wireless backbone networks through the use of backbone routers. The operation of these networks is not included in the standard.

More information on backbone networks and their implied characteristics can be found in Annex E.

## 6 System management

### 6.1 General

#### 6.1.1 Overview

System management supports network management of the network as a whole, as well as device management of the devices operating within the network. Network management includes management of the various communications resources across the network and across all layers of the architecture. Device management supports localized management of the communications resources of a device.

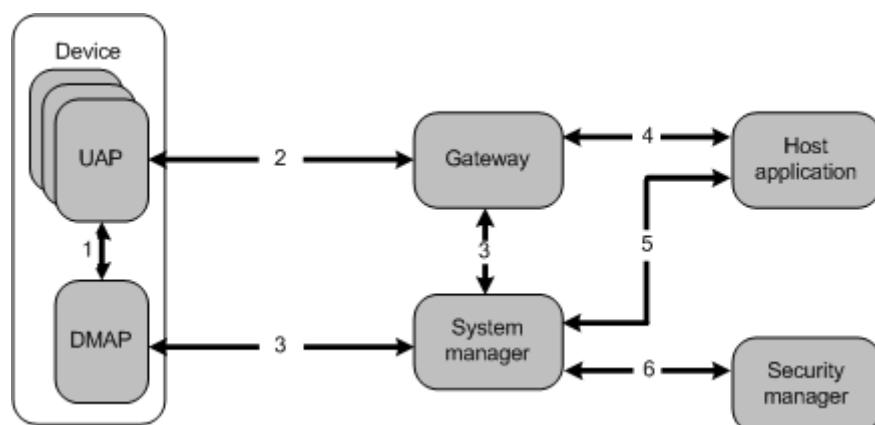
The management functions described by this standard support:

- Joining the network and leaving the network.
- Reporting of faults that occur in the network.
- Communication configuration.
- Configuration of clock distribution and the setting of system time.
- Device monitoring.
- Performance monitoring and optimization.
- Security configuration and monitoring.

#### 6.1.2 Components and architecture

The primary components of the management service include a device management application process (DMAP) that shall reside on every device compliant with this standard, as well as a system management application process (SMAP) that shall reside on a device that implements the system manager role. Roles are described in 5.2.6.3. The DMAP is a special type of user application process (UAP) that is dedicated to managing the device and its communications services. UAP is described in 12.4.3. The DMAP and the system manager shall be capable of communicating with each other over the network using the standard-defined application sub-layer services, and shall together provide a means to access management information remotely and to manage the system and its devices. System management is accomplished via inter-device messaging and device management by local intra-device communications.

The management architecture of this standard is shown in Figure 23.



**Figure 23 – Management architecture**

Devices compliant with this standard shall be managed through two distinct application processes, UAPs and the DMAP. The UAPs are configured and monitored by host applications, such as automated management systems, or by host proxy applications in the gateways. The DMAP in the device shall be managed by the system manager.

Figure 23 shows the management model relationships of this standard. For the paths illustrated in Figure 23, this standard provides a normative description of the communication

protocols for paths (2) and (3). Communication protocols for paths (1), (4), (5) and (6) are informative examples of implementations in this standard.

This standard defines the system management communication protocols that shall be used to control and monitor the DMAPs in the network and the communication paths used. These communications travel on path (3) in Figure 23.

The system manager includes communication paths outside of the standard-compliant network that allow other devices to interact with it. In Figure 23, path (5) shows a connection between the system manager and the host application. This path enables the host application to retrieve network status and request network services. The system manager also communicates with the security manager over path (6) to configure security in the network and to report status.

The user applications on devices compliant with this standard communicate with gateways and host applications using the standard protocols shown over path (2) in Figure 23. This is described in Clause 12 and Clause 13.

The plant-based host application communicates with the gateway using plant protocols that travel over path (4). The system manager does not communicate directly with the UAPs. There is an intra-device communication path that enables the DMAP and UAP processes to interact via the intra-device communication path (1) in Figure 23. This is performed over a virtual interface (1) in Figure 23, UAPME-SAP, using the UAP management object (UAPMO) which is described in 12.15.2.2.

### **6.1.3 Management functions**

Every network that is compliant with this standard shall include at least one system management role and one security management role. These roles shall be accessible to all standard complaint devices on this network.

System management is a specialized role that governs the network, the operation of devices on the network, and network communications. The functions defined within this role are performed by the system manager, providing policy-based control of the runtime communication configuration. The system manager monitors and reports on communication configuration, performance, fault conditions, and operational status. This is described in 6.3.7. The system manager also provides time-related services. Some system management functions may be completely automated, while others may be human-assisted.

The system manager supports configuration of the standard-compliant network, including attributes of the protocol suite from DL to AL for system management applications. It manages the establishment, modification and termination of contracts that are used by devices compliant with this standard to communicate with each other. The functions of the system manager do not include the control, configuration, and monitoring of the UAPs on the device. These management functions are controlled by host applications on plant networks or in handheld maintenance tools.

Security management of the system is a specialized function that is realized in one entity and that works in conjunction with the system management function to enable secure system operation. This function is performed by the security manager. Some system security management functions may be completely automated, while others may be human-assisted.

Every device compliant with this standard shall contain a DMAP. The DMAP includes a local device security management function. The DMAP cooperates with system manager and the security manager to enable the usage of system resources by the device and the secure management of the resources of a device. For example, the DMAP may ask to join the network, ask for communication bandwidth, request a communication configuration, and report its health. The system manager and the security manager authorize the device to join the network, allocate communication bandwidth, configure the device, and collect health reports. These health reports are stored in the system manager and are used to make communication configuration decisions.

In order to compartmentalize security functions, the management architecture defined by this standard supports separable system management and security management functions at both

the system and device levels. Thus, the security manager is logically separable from the system manager. More details about security manager are provided in Clause 7.

NOTE The system management and security management functions may be combined into a single entity in specific implementations.

## **6.2 Device management application process**

### **6.2.1 General**

The DMAP is a special type of application process dedicated to managing the standard-compliant device and its communications services. A DMAP resides on every device compliant with this standard.

### **6.2.2 Architecture of device management**

As shown in Figure 16, the protocol suite structure of a device includes the networking protocol layers and the user application processes.

The DMAP is shown in relation to the other protocol suite components on the right side of the protocol suite structure, including arrows depicting access to the management service access points (SAPs) for several of the protocol layers. The components within the DMAP are modeled as objects, known as management objects, which have features that are accessible over the network. The DMAP, like all application processes, is able to use the application sub-layer to communicate. The DMAP shall use the application-sub-layer SAP ASLDE-0 SAP for normal data communications. This application sub-layer SAP shall correspond to transport layer SAP TDSAP-0 which shall correspond to port number 0xF0B0. The application sub-layer provides communication services to enable the objects within the DMAP to interact with the system manager over the network. These communication services are described in 12.17.

### **6.2.3 Definition of management objects**

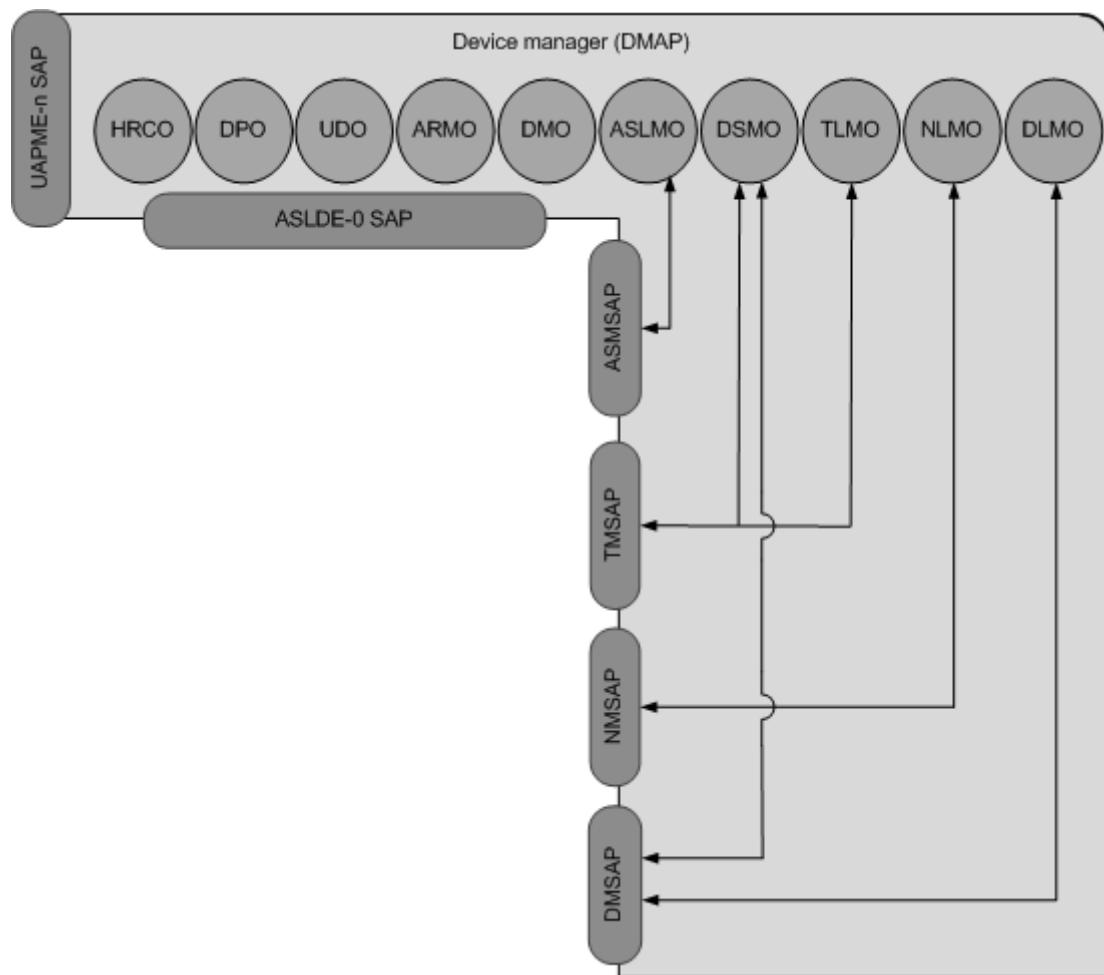
The objects defined in the DMAP follow the specification that is used to define UAP objects. The templates for defining object types, object attributes, object methods and object alerts are specified in Annex I.

The management objects are extensible by device manufacturers and network protocol suite/device developers. This is described in 12.5. Attribute and method identification space is set aside for manufacturer-defined device-specific objects. The system manager shall not be required to implement support for proprietary extensions for that device to interoperate and perform its primary function.

### **6.2.4 Management objects in device management application process**

The DMAP shall contain a number of management objects that support device management operations. These objects shall collectively perform two types of device management functions. First, these objects shall manage the device locally by manipulating attributes and invoking methods on layer management SAPs. Second, the management objects shall be accessible remotely using the ASL services such that a system manager may manipulate attributes and invoke methods on the device management objects or capture alerts from the objects. These objects are conceptual in that there are no object-oriented implementation requirements in the device, except that the externally visible behavior in terms of over-the-air ASL messaging shall be consistent with the model of object communications having the specified attributes, methods, and alerts.

As shown in Figure 24, the DMAP shall include a set of layer management objects, a device management object, a device security management object, an alert reporting management object, an upload/download object, and other management objects.



**Figure 24 – DMAP**

The standard management objects defined in this standard are given in Table 1.

**Table 1 – Standard management object types in DMAP**

<b>Defining organization: ISA</b>			
<b>Standard object type name</b>	<b>Standard object type identifier</b>	<b>Standard object identifier</b>	<b>Object description</b>
Device management object (DMO)	127	1	This object facilitates the management of the general device-wide functions of the device; see 6.2.7.1
Alert reporting management object (ARMO)	126	2	This object facilitates the management of the alert reporting functions of the device; see 6.2.7.2
Device security management object (DSMO)	125	3	This object facilitates the management of the security functions of the device; see 6.2.7.5
Data link layer management object (DLMO)	124	4	This object facilitates the management of the data link layer of the device; see 6.2.8.2.2
Network layer management object (NLMO)	123	5	This object facilitates the management of the network layer of the device; see 6.2.8.2.3
Transport layer management object (TLMO)	122	6	This object facilitates the management of the transport layer of the device; see 6.2.8.2.4
Application sub-layer management object (ASLMO)	121	7	This object facilitates the management of the application sub-layer of the device; see 6.2.8.2.6
Upload/download object (UDO)	3	8	This object facilitates the management of the upload/download functions of the device; see 6.2.7.3
Device provisioning object (DPO)	120	9	This object facilitates the provisioning of the device before it joins the network; see 6.2.7.6
Health reports concentrator object (HRCO)	128	10	This object facilitates the periodic publication of device health reports to the system manager; see 6.2.7.7
Reserved for future revisions	119-114	—	—

### **6.2.5 Communications services provided to device management objects**

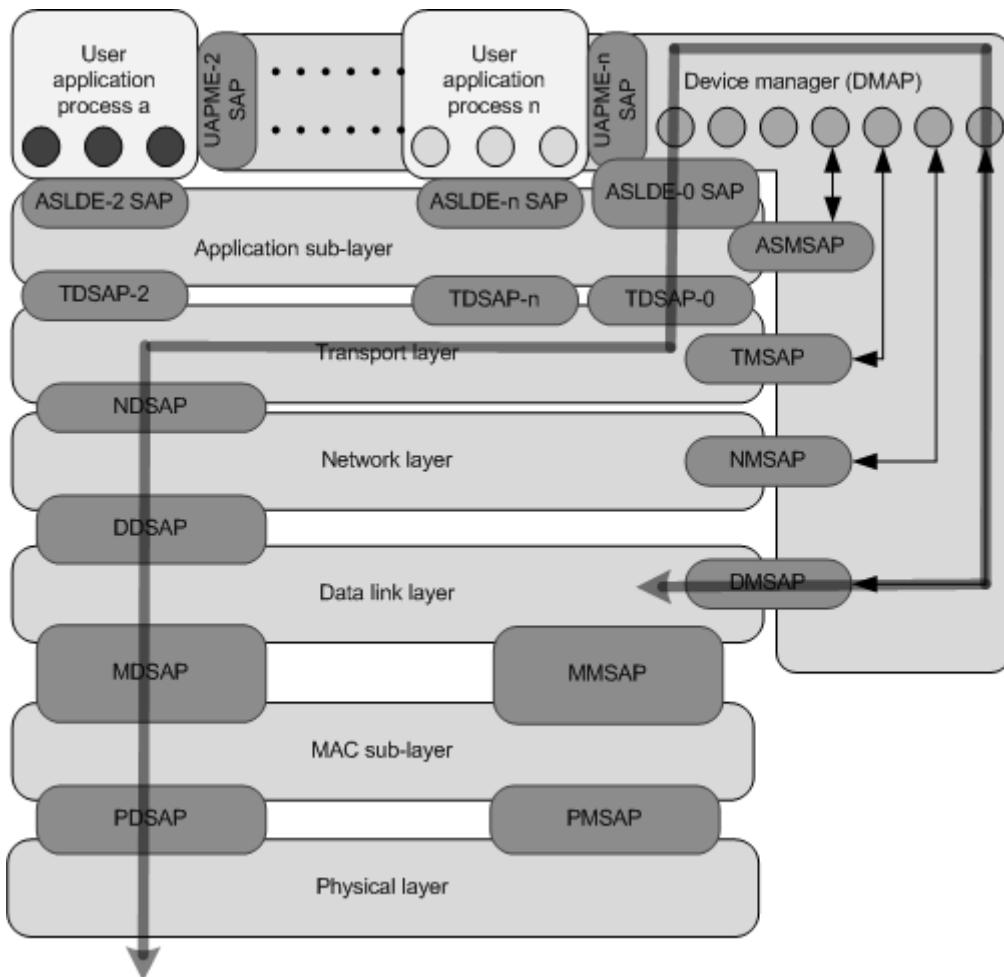
The application level services provided to the DMAP objects are the same as those provided by the application sub-layer to UAP objects. These services include client-server, publish-subscribe, source-sink, and alert reporting. Details of these services provided by the application sub-layer are given in 12.17.

As shown in Figure 25, TDSAP-0, which corresponds to port number 0xF0B0, shall be used for accessing the management objects in the DMAP, which in turn access the layer management attributes through the layer management SAP.

Access to the device management objects is protected by the transport layer security mechanisms described in 11.3.

Access to the DMAP objects is restricted to the SMAP with the following exceptions:

- 1) The ARMO can also be accessed by alert masters that receive alerts originating in the device. See 6.2.7.2.3.
- 2) A joining device is allowed to access the device management object methods used during the join process. See 6.3.9.2.2.



**Figure 25 – Example of management SAP flow through standard protocol suite**

Client-server interactions (including reading and writing of attributes, executing of methods, joining, and requesting contracts) are the primary tools used for system management. In addition, the DMAP may use the alert reporting services of the ASL to report to the system manager when certain management-related conditions are detected. Designers of management objects may use alerts at various priority levels to help accomplish system and device management functions.

## 6.2.6 Attributes of management objects

### 6.2.6.1 General

The layer management SAPs shown in Figure 25 provide access to the management information in the various layers of the protocol suite.

This information is represented by attributes defined in the management objects of the DMAP that can be monitored and operated on by the system manager. Details of the management objects are given in 6.2.3. The attributes in the layer management objects are used to configure the protocol layers and to monitor their status. The template for describing the attributes in all the management objects is provided in Annex I.

Attributes shall either have a data type that is a standard-defined scalar type or a standard-defined data structure. More details about attributes are given in 12.6.2.

A structured attribute is a special type of attribute that shall have a data type that is an array of standard-defined data structures. Management information that needs to be visualized as a collection of one or more tables is modeled as structured attributes defined in the management objects.

Attributes defined in the management objects can be accessed using the standard ASL-provided read or write services. Such operations enable configuration of the layers, as well as monitoring of their status. They can be used to retrieve, set / modify, and reset the values of attributes. Operations on attributes are described in Annex J.

### **6.2.6.2 Structured attribute index field**

As structured attributes are described as arrays of data structures, one or more index fields for such arrays need to be indicated in the definition of each such structured attribute. This is done by including an \* (asterisk) after the element name(s) in the table describing the data structure. The template for defining a data structure is given in Annex I.

### **6.2.6.3 Metadata of structured attribute**

Structured attributes represent information tables. In order to provide external access to the statistics of such tables, attributes may be defined for each of the management objects that contain such structured attributes. Such attributes represent the metadata of the corresponding structured attributes.

The standard data type for a metadata attribute is given in Table 2.

**Table 2 – Meta\_Data\_Attribute data structure**

<b>Standard data type name: Meta_Data_Attribute</b>		
<b>Standard data type code: 406</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
Count (number of indexed rows currently in the attribute)	1	Type: Unsigned16 Classification: Static Accessibility: Read only Default value : N/A Valid value set: 0 – 0xFFFF
Capacity (number of rows that the attribute can hold)	2	Type: Unsigned16 Classification: Static Accessibility: Read only Default value : N/A Valid value set: 0 – 0xFFFF

## **6.2.7 Definitions of management objects in device management application process**

### **6.2.7.1 Device management object**

As shown in Figure 24, the DMAP includes a set of management objects. The device management object (DMO) in the DMAP shall provide access to attributes having device-wide scope. Attributes of the DMO shall include the EUI-64, vendor ID, serial number, communications software revision, and power supply status of the device. More details about DMO are provided in 6.2.8.1.

### **6.2.7.2 Alert reporting management object**

#### **6.2.7.2.1 General**

The alert reporting management object (ARMO) shall be used to manage all of the alert reports for the device. Alert is the term used to describe the action of reporting an event condition or an alarm condition. Event is the term for a stateless condition, reporting when something happened. Alarm is the term used for a condition that maintains state; the alarm condition remains true until the alarm condition clears. Alerts, including events and alarms, are envisioned to be of high utility for managing a network compliant with this standard.

There shall be at most one ARMO per device. Both alarms and events shall be reported through the ARMO. When an alert is triggered, it indicates a significant situation that needs to be reported. The ARMO shall encapsulate the report, handle timeouts and retries, and throttle alert reporting from the device.

The ARMO functions as an alert proxy for the objects present in the device. All alerts generated by any object present in a device shall be sent only by the ARMO which is a management object that is part of the DMAP. The alert data packet shall indicate in the APDU

header that the originator of the communication is the ARMO object and the DMAP of the device reporting the alert. The object and UAP that originated the actual alert packet shall be identified in the content of the alert report and not in the APDU headers.

Each alert shall be acknowledged by the device receiving the alert. Each alert acknowledgment shall be addressed to the ARMO of the device that originated the alert. Alerts are reported promptly and time-stamped accurately using queued alert reporting. Queued alert reporting involves the alert detecting device reporting the condition using a source/sink communication flow and receiving an acknowledgement in return.

**NOTE** The intent of specifying the ARMO in this standard is to separate alert detection from the management of reporting the alert condition. Unlike some wire-oriented legacy protocols, this standard consolidates alerts locally in order to minimize externalized messaging and energy consumption.

The alert model used in this standard is described in 12.8.

The interfaces between the ARMO and all other objects, both in UAPs and in the DMAP, are device internal and are not specified in this standard.

#### 6.2.7.2.2 Alert types

Alert classes, alert directions, and alert priorities are defined in 12.11. The alert category indicates whether the alert is a device diagnostic alert, a communication diagnostic alert, a security alert, or a process alert. The alert type provides additional information regarding the alert, specific to the alert category and specific to the application object generating the alert.

Table 3 provides the alert types for the alert categories of communication diagnostic alert category. Table 4 provides the alert types for the security alert category. Table 5 provides the alert types for the device diagnostic alert category. Table 6 provides the alert types for the process alert category.

**Table 3 – Alert types for communication diagnostic category**

Alert type	Alert category: Communication diagnostic					
	ARMO	ASLMO	DLMO	NLMO	TLMO	DMO
0	Alarm_Recovery_Start; see Table 8	MalformedAPDU CommunicationAlert; see 12.19.5	DL_Connectivity; see 9.6.1	NL_Dropped PDU; see 10.4.3	IllegalUseOfPort; see 11.6.2.5.4	Device_Power_Status_Check; see 6.2.8.1.2
1	Alarm_Recovery_End; see Table 8		NeighborDiscovery; see 9.6.2		TPDUonUnregisteredPort; see 11.6.2.5.4	Device_Restart; see 6.2.8.1.2
2					TPDUoutOfSecurityPolicies; see 11.6.2.5.4	

**Table 4 – Alert types for security alert category**

Alert type	Alert category: Security		DPSO
	ARMO	DSMO	
0	Alarm_Recovery_Start; see Table 8	Security_MPDU_Fail_Rate_Exceeded; see 7.11.4	Not_On_Whitelist_Alert; see Table 427
1	Alarm_Recovery_End; see Table 8	Security_TPDU_Fail_Rate_Exceeded; see 7.11.4	Inadequate_Join_Capability_Alert; see Table 427
2		Security_Key_Update_Fail_Rate_Exceeded; see 7.11.4	

**Table 5 – Alert types for device diagnostic alert category**

Alert type	Alert category: Device diagnostic	
	ARMO	
0	Alarm_Recovery_Start; see Table 8	
1	Alarm_Recovery_End; see Table 8	

**Table 6 – Alert types for process alert category**

Alert type	Alert category: Process				
	ARMO	AI	AO	BI	BO
0	Alarm_Recovery_Start; see Table 8	See 12.19.7	See 12.19.7	See 12.19.7	See 12.19.7
1	Alarm_Recovery_End; see Table 8				

#### **6.2.7.2.3 Alert master**

Alerts shall be sent to alert receiving objects. Alert receiving objects are defined in 12.15.2.3. Each alert category may have a different alert receiving object residing in a different device. Devices that receive these alerts are known as alert masters.

DMAP access is typically restricted to the SMAP present in the system manager. In an exception to this general principle, alert masters are allowed to access the ARMO object present in the DMAP. DMAP access by alert masters shall be limited to the ARMO, unless the alert master uses the DMAP-SMAP session established when the device joined the network. The alert masters to which the device is configured to send alerts are listed in Table 7.

#### **6.2.7.2.4 Alert queue**

Alerts belonging to each category are assumed to be placed into an internal queue provided per-category in the device. Both types of alerts, events (stateless) and alarms (stateful) will be placed in the same queue, filtered by category. The queue is necessary to provide a guaranteed delivery of alerts to the alert master. Every alert to be reported to the alert master is placed into this reporting queue.

The size of the queue should be big enough to accommodate all events as well as all possible alarm conditions simultaneously in order to support alarm recovery without losing any alarms.

Although placed in the same queue, events and alarms will be prioritized differently. The device shall report an event with higher priority before an event with lower priority. For alarms, the queue is emptied sequentially; the oldest alarm is reported first. When the queue is full and a new alarm is submitted, the oldest alarm is dropped from the queue regardless of its reporting state.

#### **6.2.7.2.5 Alert state models**

The state tables and transitions for alarms and events are given in 12.9 and 12.10.

#### **6.2.7.2.6 Alarm recovery**

It is often useful to be able to recover all alarms currently active within a device. The need for alarm recovery arises whenever a connection to device is lost for a period of time or whenever an alert master commands an alarm recovery.

Alarm recovery consists of the following set of activities:

- The alert master shall command an alarm recovery by using the Alarm\_Recovery method of the ARMO. This method is described in Table 9.
- The ARMO shall send a recovery start alert to the alert master. This alert indicates that the ARMO has received a command to recover alarms and that active alarms will follow (the process for re-sending these active alarms within a device is not specified).
- The ARMO shall send a recovery end alert to the alert master.

The ARMO is responsible for generating alarm recovery start and end alerts and for coordinating the alarm recovery process with the application objects residing within the device.

#### **6.2.7.2.7 Alert reporting management object attributes, alerts and methods**

The attributes of the ARMO are defined in Table 7.

**Table 7 – ARMO attributes**

<b>Standard object type name: Alert reporting management object (ARMO)</b>				
<b>Standard object type identifier: 126</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Alert_Master_Device_Diagnostics	1	Alert master for alerts that belong to the device diagnostics category	Type: Alert communication endpoint Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: See data type definition in 12.16.2.5	Typically set to be gateway of the device, but can be changed to any other standard-compliant device with valid 128-bit address; this information is expected to be configured by the host application after the device joins the network.
Confirmation_Timeout_Device_Diagnostics	2	Timeout waiting for acknowledgment of a device diagnostic alert that was sent to the alert master	Type: Integer16 Classification: Static Accessibility: Read/write Initial default value: 10 Valid value set: -32768 to + 32767	Timeout independent of proximity to alert master; Valid value set: a positive number is a multiple of 1 s and a negative number is a fraction of 1 s.
Alerts_Disable_Device_Diagnostics	3	Command to disable / enable all device diagnostic alerts	Type: Boolean Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 0-1	1= disable 0= enable
Alert_Master_Comm_Diagnostics	4	Alert master for alerts that belong to the communication diagnostics category	Type: Alert communication endpoint Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: See data type definition in 12.16.2.5	Typically set to be system manager of the device, but can be changed to any other standard-compliant device with valid 128-bit address; the device shall set this to be its system manager after it joins the network; see 6.3.7.2
Confirmation_Timeout_Comm_Diagnostics	5	Timeout waiting for acknowledgment of a communication diagnostic alert that was sent to the alert master	Type: Integer16 Classification: Static Accessibility: Read/write Initial default value: 10 Valid value set: -32768 to + 32767	Timeout independent of proximity to alert master; Valid value set: a positive number is a multiple of 1 s and a negative number is a fraction of 1 s.
Alerts_Disable_Comm_Diagnostics	6	Command to disable / enable all communication diagnostic alerts	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 0-1	1= disable 0= enable
Alert_Master_Security	7	Alert master for alerts that belong to the security	Type: Alert communication	Typically set to be system manager of

<b>Standard object type name: Alert reporting management object (ARMO)</b> <b>Standard object type identifier: 126</b> <b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
		category	endpoint Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: See data type definition in 12.16.2.5	the device, but can be changed to any other standard-compliant device with valid 128-bit address; the device shall set this to be its system manager after it joins the network; see 6.3.7.2
Confirmation_Timeout_Security	8	Timeout waiting for acknowledgment of a security alert that was sent to the alert master	Type: Integer16 Classification: Static Accessibility: Read/write Initial default value: 10 Valid value set: -32768 to + 32767	Timeout independent of proximity to alert master; Valid value set: a positive number is a multiple of 1 s and a negative number is a fraction of 1 s.
Alerts_Disable_Security	9	Command to disable / enable all security alerts	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 0-1	1= disable 0= enable
Alert_Master_Process	10	Alert master for alerts that belong to the process category	Type: Alert communication endpoint Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: See data type definition in 12.16.2.5	Typically set to be gateway of the device, but can be changed to any other standard-compliant device with valid 128-bit address; this information is expected to be configured by the host application after the device joins the network.
Confirmation_Timeout_Process	11	Timeout waiting for acknowledgment of a process alert that was sent to the alert master	Type: Integer16 Classification: Static Accessibility: Read/write Initial default value: 10 Valid value set: -32768 to + 32767	Timeout independent of proximity to alert master; Valid value set: a positive number is a multiple of 1 s and a negative number is a fraction of 1 s.
Alerts_Disable_Process	12	Command to disable / enable all process alerts	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 0-1	1= disable 0= enable
Comm_Diagnostics_Alarm_Recovery_AlertDescriptor	13	Used to change the priority of alarm recovery start and end events (described in Table 8) that belong to the comm.	Type: Alert report descriptor Classification: Static	

<b>Standard object type name: Alert reporting management object (ARMO)</b>				
<b>Standard object type identifier: 126</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
		diagnostics category; these events can also be turned on or turned off	Accessibility: Read/write  Initial default value: Alert report disabled = False Alert report priority = 3  Valid value set: See type definition in 12.16.2.7	
Security_Alarm_Recovery_AlertDescriptor	14	Used to change the priority of alarm recovery start and end events (described in Table 8) that belong to the security category; these events can also be turned on or turned off	Type: Alert report descriptor  Classification: Static  Accessibility: Read/write  Initial default value: Alert report disabled = False Alert report priority = 3  Valid value set: See type definition in 12.16.2.7	
Device_Diagnostics_Alarm_Recovery_AlertDescriptor	15	Used to change the priority of alarm recovery start and end events (described in Table 8) that belong to the device diagnostics category; these events can also be turned on or turned off	Type: Alert report descriptor  Classification: Static  Accessibility: Read/write  Initial default value: Alert report disabled = False Alert report priority = 3  Valid value set: See type definition in 12.16.2.7	
Process_Alarm_Recovery_AlertDescriptor	16	Used to change the priority of alarm recovery start and end events (described in Table 8) that belong to the process category; these events can also be turned on or turned off	Type: Alert report descriptor  Classification: Static  Accessibility: Read/write  Initial default value: Alert report disabled = False Alert report priority = 3  Valid value set: See type definition in 12.16.2.7	
Reserved for future standard releases	17-63	—	—	—

The alerts of the ARMO are defined in Table 8.

**Table 8 – ARMO alerts**

Standard object type name(s): Alert reporting management object (ARMO)					
Standard object type identifier: 126					
Defining organization: ISA					
Description of the alert: Alarm recovery begin and end events for alarms that belong to all categories					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority (Enumerated: urgent, high, med, low, journal)	Value data type	Description of value included with alert
0 = Event	1 = Comm. diagnostics	0 = Alarm_Recovery_Start	3 = Low	Type: N/A	Generated by ARMO for the comm. diagnostics alert master indicating that the alarm recovery command has been received; all outstanding comm. diagnostic alarms are reported after this event is raised
				Initial default value: N/A	
				Valid value set: N/A	
0 = Event	1 = Comm. diagnostics	1 = Alarm_Recovery_End	3 = Low	Type: N/A	Generated by ARMO for the comm. diagnostics alert master indicating that the alarm recovery process has ended
				Initial default value: N/A	
				Valid value set: N/A	
0 = Event	2 = Security	0 = Alarm_Recovery_Start	3 = Low	Type: N/A	Generated by ARMO for the security alert master indicating that the alarm recovery command has been received; all outstanding security alarms are reported after this event is raised
				Initial default value: N/A	
				Valid value set: N/A	
0 = Event	2 = Security	1 = Alarm_Recovery_End	3 = Low	Type: N/A	Generated by ARMO for the security alert master indicating that the alarm recovery process has ended
				Initial default value: N/A	
				Valid value set: N/A	
0 = Event	0 = Device diagnostics	0 = Alarm_Recovery_Start	3 = Low	Type: N/A	Generated by ARMO for the device diagnostics alert master indicating that the alarm recovery command has been received; all outstanding device diagnostic alarms are reported after this event is raised
				Initial default value: N/A	
				Valid value set: N/A	
0 = Event	0 = Device diagnostics	1 = Alarm_Recovery_End	3 = Low	Type: N/A	Generated by ARMO for the device diagnostics alert master indicating that the alarm recovery process has ended
				Initial default value: N/A	
				Valid value set: N/A	
0 = Event	3 = Process	0 = Alarm_Recovery_Start	3 = Low	Type: N/A	Generated by ARMO for the process alert master indicating that the alarm recovery command has been received; all outstanding process alarms are reported after this event is raised
				Initial default value: N/A	
				Valid value set: N/A	
0 = Event	3 = Process	1 = Alarm_Recovery_End	3 = Low	Type: N/A	Generated by ARMO for the process alert master indicating that the alarm recovery process has ended
				Initial default value: N/A	
				Valid value set: N/A	

The method of the ARMO used to recover alarms of the different categories shall be as defined in Table 9.

**Table 9 – Alarm\_Recovery method**

<b>Standard object type name(s): Alert reporting management object (ARMO)</b>			
<b>Standard object type identifier: 126</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Alarm_Recovery	1	Method to recover alarms that belong to the category mentioned in the input argument	
	<b>Input arguments</b>		
	Argument #	Argument name	Argument type (data type and length)
	1	Alert_Category	Data type: Unsigned8 Enumeration: 0 = device diagnostics 1 = comm. diagnostics 2 = security 3 = process
	<b>Output arguments</b>		
	Argument #	Argument name	Argument type (data type and length)
	-		

### 6.2.7.3 Upload/download object

The attributes, methods and state machines of the UDO in the DMAP shall be as per the definition given in 12.15.2.4. The object identifier of the UDO in the DMAP shall be 8.

An upload/download object (UDO) is used for uploading or downloading large blocks of information to/from a device. The UDO may be used to support downloading a new version of communications firmware or data. The UDO maintains revision control information. UDO is described in 12.15.2.4.

The firmware upgrade process used by the system manager for over-the-air firmware upgrades is described in 6.3.6. The methods and attributes of the upload/download object in the DMAP of the device can be used for sending firmware updates to the device.

The firmware upgrade process may include a cut-over mechanism that specifies a cut-over time (after the update is delivered), at which point devices begin using the new firmware. The CutoverTime attribute in the UDO shall be used to indicate this cut-over time. The cut-over time uses the shared sense of time configured by the system manager.

Support is provided for vendor-specific, device model-specific, and device instance-specific updates. Before cut-over, the upload/download object of the device may perform safety checks on a received update to assure that an update is appropriate for a specific device type. As all the communication takes place between application objects, such an update is protected by the end to end transport layer security mechanism. In addition, the firmware update may use security mechanisms to authenticate the update. As part of the firmware upgrade process, the upload/download object of the device may be provided with the appropriate standardized labeling, versioning, and security information for these verifications by the host update application. These verifications may be vendor-specific and are not specified by this standard.

NOTE The UDO in the DMAP to update firmware is described here. Details about general upload/download objects that may be present in general application processes are given in 12.15.2.4.

#### **6.2.7.4 Layer management objects**

The set of objects within the DMAP shall include objects representing access to each of the layer management SAPs. These objects include:

- The application sub-layer management object (ASLMO), which provides access to the ASMSAP.
- The transport layer management object (TLMO), which provides access to the TMSAP.
- The network layer management object (NLMO), which provides access to the NMSAP.
- The data link management object (DLMO), which provides access to the DMSAP.

The services, defined by the layer SAP definitions, are reflected in the features of the management objects such that, effectively, the services made available at the management SAPs become remotely accessible using secure standard communications mechanisms. Generically, the management SAPs provide for reading and writing attributes, invoking methods, and reporting events. The various layer specifications specify the exact features that are available on each of these SAPs. In effect, these specifications define the layer management objects. See 6.2.8.2 for more details on the layer management objects.

#### **6.2.7.5 Device security management object**

The DMAP shall include a device security management object (DSMO) that provides appropriately limited access to device security management functions. The DSMO manages security key material and cryptographic operations. The details of this object are provided in 7.11.

#### **6.2.7.6 Device provisioning object**

The DMAP shall include a device provisioning object (DPO) that is accessed during the provisioning process of the device. More details about the attributes, methods and alerts of the DPO are provided in 14.9.

#### **6.2.7.7 Health reports concentrator object**

The DMAP shall include a health reports concentrator object (HRCO) that can be configured by the system manager to enable periodic publication of device health reports. The device health reports may consist of periodic publication of one or more attributes from the management objects in the DMAP.

The attributes of the HRCO are as per the definition of the concentrator object given in 12.15.2.5. The object identifier of the HRCO in the DMAP shall be 10. The CommunicationEndPoint and Array of ObjectAttributeIndexAndSize attributes of the HRCO are used by the system manager to set up periodic publications of health reports from the device. The system manager may choose to include any attribute from any management object in such health reports which are used for system performance monitoring. System performance monitoring is described in 6.3.7.

### **6.2.8 Functions of device management and layer management**

#### **6.2.8.1 Device management functions**

##### **6.2.8.1.1 General**

Device management capabilities are provided primarily via access to DMO attributes and invocation of methods. The DMO contains critical attributes of device-wide scope that shall be available in all devices. Product implementers may extend the list of attributes beyond the required attributes described below.

Some attributes available via the DMO may also be available as an attribute of a particular layer management object. In that case, change in the value of any such attribute shall be reflected in the corresponding attribute.

The DMO shall provide system time information to other management objects; the DMO may obtain this system time information by interacting with the DL of the device and / or the system manager or from another source, such as a GPS receiver within the device. Time-

keeping by the DL is described in the 9.1.9. The role of the system manager in maintaining time across the network is described in 6.3.10.

The establishment, modification and termination of contracts for a device shall be managed by its DMO. Contracts are described in 6.3.11.2.

The attributes of the DMO are defined in Table 10.

**Table 10 – DMO attributes**

<b>Standard object type name: Device management object (DMO)</b>				
<b>Standard object type identifier: 127</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
EUI64	1	64-bit unique identifier of device	Type: Unsigned64	This shall be a global unique IEEE EUI-64 address; this attribute is a duplicate of corresponding attributes in DLMO and NLMO.
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
DL_Address_16_Bit	2	16-bit DL address of device in its DL subnet	Type: Unsigned16	Address unique in DL subnet of device; assigned by system manager; address 0 means that the device does not have an operational DL; this attribute is a duplicate of corresponding attributes in DLMO and NLMO; The range of unicast addresses shall be limited to the range of 1-32767. 0 indicates no address, and ranges above 32767 are reserved for multicast addresses, graph IDs, and other future purposes. This attribute shall be configured by the system manager during the device join process.
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0	
			Valid value set: 0-32767	
Network_Address_128_Bit	3	128-bit network address assigned by system manager	Type: Unsigned128	Network address unique in network of device and used by application to identify devices across the network; this attribute is a duplicate of corresponding attributes in DLMO and NLMO; Value 0 means that an address has not yet been assigned. This attribute shall be configured by the system manager during the device join process.
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0	
			Valid value set: 0 – 2 <sup>127</sup> -1	
Device_Role_Capability	4	Role(s) that the device is capable of playing in the network; roles are defined in 5.2.6.2	Type: Unsigned16	This attribute shall be sent to the system manager during the device join process; see 6.3.9.2. See 5.2.6.2 for acceptable roles and their descriptions. Bit map with the following mapping (bit value: 0 – no, 1 – yes): Bit 0 – IO Bit 1 – router Bit 2 – backbone router Bit 3 – gateway Bit 4 – system manager Bit 5 – security
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	

				manager Bit 6 – system time source Bit 7 – provisioning device Bits 8 to 15 – reserved (reserved bits shall be set to 0)
Assigned_Device_Role	5	Role(s) of the device as assigned by the system manager; roles are defined in 5.2.6.2	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A	This attribute shall be sent to the device by the system manager during the device join process; see 6.3.9.2. Refer to 5.2.6.2 for acceptable roles and their descriptions. This attribute shall be configured by the system manager during the device join process. The assigned role for a device shall not exceed its capabilities. Bit map with the following mapping (bit value: 0 – no, 1 – yes): Bit 0 – IO Bit 1 – router Bit 2 – backbone router Bit 3 – gateway Bit 4 – system manager Bit 5 – security manager Bit 6 – system time source Bit 7 – provisioning device Bits 8 to 15 – reserved (reserved bits shall be set to 0)
Vendor_ID	6	Human-readable identification of device vendor	Type: VisibleString Classification: Constant Accessibility: Read only Initial default value: N/A Valid value set: N/A	Assigned by vendor during device manufacturing; max length: 16 octets
Model_ID	7	Human-readable identification of device model	Type: VisibleString Classification: Constant Accessibility: Read only Initial default value: N/A Valid value set: N/A	Assigned by vendor during device manufacturing; max length: 16 octets
Tag_Name	8	Tag name of device	Type: VisibleString Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A	Assigned by user; max length: 16 octets
Serial_Number	9	Serial number of device	Type: VisibleString Classification: Constant Accessibility: Read only Initial default value: N/A Valid value set: N/A	Assigned by vendor during device manufacturing; max length: 16 octets
Power_Supply_Status	10	Status information of power supply of device	Type: Unsigned8 Classification: Dynamic Accessibility: Read/write Initial default value: N/A Valid value set: 0-3	Enumeration: 0 – line powered 1 – battery powered, greater than 75% remaining capacity 2 – battery powered, between 25% and 75% remaining capacity 3 – battery powered,

				less than 25% remaining capacity
Device_Power_Status_Check_AlertDescriptor	11	Used to change the priority of Device_Power_Status_Check alert (described in Table 11; this alert can also be turned on or turned off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Initial default value: Alert report disabled = False Alert report priority = 8 Valid value set: See type definition	
DMAP_State	12	Status of DMAP	Type: Unsigned8 Classification: Dynamic Accessibility: Read only Initial default value: 1 Valid value set: 0-2	DMAP state diagram is same as UAP state diagram given in 12.15.2.2.3 0 = Inactive 1 = Active 2 = Failed
Join_Command	13	Command informing device to join the system, restart itself and re-join, or reset to factory defaults	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 0-4	The use of this attribute is described in 6.3.9. The value None (0) shall not be indicated in a write request. Only the provisioning device is expected to be able to issue the Join_Command = 1 as the device has not yet joined the network and so is not accessible by any other device. Warm restart shall preserve static and constant attributes data including contracts and session keys. Restart as provisioned corresponds to the provisioned state of the device in which the device only retains the information that is received during its provisioning step. Reset to factory defaults corresponds to the unconfigured device phase in Figure 5. Enumeration: 0 = none 1 = join / start 2 = warm restart 3 = restart as provisioned 4 = reset to factory defaults
Static_Revision_Level	14	Revision level of the static data associated with all management objects	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Initial default value: 0 Valid value set: 0 – 0xFFFFFFFF	Revision level is incremented each time a static attribute value in any management object is changed; value rolls over when limit is reached; value resets whenever the device is reset to factory defaults (join command = 4).
Restart_Count	15	Number of times device restarted	Type: Unsigned16 Classification: Static Accessibility: Read only Initial default value: 0 Valid value set: 0-0xFFFF	Device restart can be due to battery replacement, warm restart command, firmware download, link failure; value rolls over

				if max value is reached; value resets to 0 when device is reset to factory defaults.
Uptime	16	Low accuracy counter for counting seconds since last device restart	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Initial default value: 0 Valid value set: 0 – 0xFFFFFFFF	Units in seconds; reset to 0 if device restarts
Device_Memory_Total	17	Total memory of device expressed in bytes	Type: Unsigned32 Classification: Constant Accessibility: Read only Initial default value: N/A Valid value set: N/A	Units in octets
Device_Memory_Used	18	Memory currently used in device expressed in bytes	Type: Unsigned32 Classification: Dynamic Accessibility: Read only Initial default value: N/A Valid value set: N/A	Units in octets
TAI_Time	19	Current TAI time	Type: TAINetworkTimeValue Classification: Dynamic Accessibility: Read only Initial default value: 0 Valid value set: See 12.22.4.2	Value is obtained either from DL (if device is not system time source) or from backbone / external source (if device is system time source or is on the backbone and does not have a DL)
Comm_SW_Major_Version	20	Major version of communications software currently being used in the device	Type: Unsigned8 Classification: Constant Accessibility: Read only Initial default value: 0 Valid value set: 0-0xFF	8-bit communications software major version number assigned by ISA; Current value for this standard shall be 0.
Comm_SW_Minor_Version	21	Minor version of communications software currently being used in the device	Type: Unsigned8 Classification: Constant Accessibility: Read only Initial default value: 0 Valid value set: 0-0xFF	8-bit communications software minor version number assigned by ISA; Current value for this standard shall be 1.
Software_Revision_Informatio n	22	Revision information about communications software for particular major and minor version numbers	Type: VisibleString Classification: Constant Accessibility: Read only Initial default value: N/A Valid value set: N/A	Revision information assigned by vendor; Max length: 16 octets
System_Manager_128_Bit_Address	23	Network address of system manager	Type: Unsigned128 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: N/A	This information shall be provided to device either during provisioning process or during join process
System_Manager_EUI64	24	EUI-64 of system manager	Type: Unsigned64 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: N/A	This information shall be provided to device either during provisioning process or during join process
System_Manager_DL_Address_16_Bit	25	16-bit DL address of system manager in DL subnet of device	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: N/A	This attribute shall be configured by the system manager during the device join process.
Contracts_Table	26	Table that includes information about all existing	Type: Array of Contract_Data Classification: Static Accessibility: Read/write	Updated when a corresponding contract gets established, modified, renewed or

		contracts of the device	Initial default value: N/A Valid value set: See Contract_Data data type definition	terminated; see 6.3.11.2 for more details about contracts and about data type Contract_Data. A new entry in the Contracts_Table shall be created everytime a contract response associated with the successful creation of a new contract is received from the system manager. For additional details see 6.3.11
Contract_Request_Timeout	27	Timeout for DMO before the contract request can be retried	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 30 Valid value set: 0-0xFFFF	System manager sets this timeout value after the device joins the network; units in seconds
Max_ClientServer_Retries	28	The maximum number of client request retries DMAP shall send in order to have a successful client/server communication	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: 3 Valid value set: 0-8	The number of retries sent for a particular message may vary by message based on application process determination of the importance of the message. For example, some messages may not be retried at all, and others may be retried the maximum number of times
Max_Retry_Timeout_Interval	29	The maximum timeout interval for a client request before it is sent again	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 30 Valid value set: 0 – 0xFFFF	System manager sets this timeout value after the device joins the network; units in seconds
DMAP_Objects_Count	30	Number of management objects in DMAP including this DMO	Type: Unsigned8 Classification: Static Accessibility: Read only Initial default value: 1 Valid value set: 1 – 255	Total count of the management objects such as DLMO, NLMO, etc in the DMAP of this device; all application processes in the device shall include an attribute with such information
DMAP_Objects_List	31	List of all the management objects in the DMAP	Type: Array of ObjectIDandType Classification: Static Accessibility: Read only Initial default value: N/A Valid value set: N/A	List to identify all the management objects that are available in the DMAP; all application processes in the device shall include an attribute with such information. See 12.16.2.10 for details about this data type
Metadata_Contracts_Table	32	Metadata (count and capacity) of the Contracts_Table attribute	Type: Meta_Data_Attribute	Metadata containing a count of the number of entries in the table and capacity (the total number of rows allowed) for the table; see 6.2.6.3 for details about this data type
Non_Volatile_Memory_Capability	33	Indicates if device is capable of maintaining all DMAP information	Type: Unsigned8 Classification: Constant Accessibility: Read only Initial default value: N/A	0 = Device is not capable of maintaining all DMAP information that falls under the

		that falls under the Static classification in non-volatile memory over a power-cycle or not	Valid value set: 0 – 1	Static classification in non-volatile memory over a power-cycle; 1 = Device is capable of maintaining all DMAP information that falls under the Static classification in non-volatile memory over a power-cycle. See 6.3.9.4.2 for more information
Warm_Restart_Attempts_Timeout	34	The timeout after which a device that is trying to re-join the network through a warm restart reverts back to the restart as provisioned state	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 60 Valid value set: 0 – 0xFFFF	Units in minutes; see 6.3.9.4.2 for more information
Device_Restart_AlertDescriptor	35	Used to change the priority of Device_Restart alert (described in Table 11; this alert can also be turned on or turned off)	Type: Alert report descriptor Classification: Static Accessibility: Read/write Initial default value: Alert report disabled = False Alert report priority = 8 Valid value set: See type definition	
Reserved for future standard releases	36-63	—	—	—

### 6.2.8.1.2 Device management object alerts

The DMO of the device shall send an alert to indicate a change in its power status. The DMO of the device shall send an alert whenever it goes through a device restart. Device restart is described in 6.3.9.4.2.

The alerts of the DMO are defined in Table 11.

**Table 11 – DMO alerts**

<b>Standard object type name(s): Device management object (DMO)</b>					
<b>Standard object type identifier: 127</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Communication diagnostic alerts to indicate that the device power supply status has changed and to indicate that the device has restarted</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority (Enumerated: urgent, high, med, low, journal)	Value data type	Description of value included with alert
0 = Event	1 = Comm. diagnostics	0 = Device_Power_ Status_Check	8 = Medium	Type: Unsigned8  Initial default value: N/A  Valid value set: 0 - 3	The current value of the Power_Supply_Status attribute in Table 10 is included in this alert
0 = Event	1 = Comm. diagnostics	1 = Device_Restart	8 = Medium	Type: N/A	Only a device that has DMO attribute Non_Volatile_Memory_Capability = 1 can send this alert to indicate that it has gone through a warm restart

#### 6.2.8.1.3 Device management object methods

The methods of the DMO are described in 6.3.9.2.2, 6.3.11.2.10.5, and 6.3.11.2.11.3.

#### 6.2.8.2 Layer management

##### 6.2.8.2.1 General

Each communication layer within the protocol suite has self-contained layer management functionality. Each layer management function provides a management SAP. Device management of layers is accomplished via access to the management SAPs on each of the layers, as shown in Figure 25. Management of the layers within a device may be done locally by functionality that resides within the DMO, since the DMO has access to the management SAPs. In addition, as discussed in 6.2.4, layer management may be accomplished remotely by a system manager.

The formal definition of each of the layer management objects is included below. The definition of the layer management objects within the DMAP corresponds directly with the layer management SAP definition. However, there may be a need to restrict remote access to specific features of a given layer management SAP. Thus, some attributes or methods, while accessible to the local DMO, should not be remotely accessible. Such restrictions, whenever necessary, are specified in the layer specifications. The operations described in Annex J can be used to access attributes in these layer management objects.

### **6.2.8.2.2 Data link layer management object**

#### **6.2.8.2.2.1 General**

In the architecture defined by this standard, all DL, MAC, and PhL layer management services are provided via a single data link management service access point (DMSAP). There is no direct access to management SAPs for the physical layer and MAC layer. DLMO provides the attributes that are accessible remotely.

#### **6.2.8.2.2.2 Physical layer management**

Physical layer management entities are manipulated indirectly via the DMSAP. DL attributes relate to a subset of those defined in IEEE Std 802.15.4, as described in 9.1.5.

#### **6.2.8.2.2.3 Media access control sub-layer management**

MAC sub-layer management entities are manipulated indirectly via the DMSAP. DL attributes relate to a subset of those defined in IEEE Std 802.15.4, as described in 9.1.5.

#### **6.2.8.2.2.4 Data link layer management**

DL management attributes and methods are available via the DMSAP. Attributes, methods, and alerts of the DLMO are defined in 9.4 and 9.6.

#### **6.2.8.2.3 Network layer management**

Network layer management attributes and methods are available via the NMSAP. Attributes, methods and alerts of the NLMO are defined in 10.4.

#### **6.2.8.2.4 Transport layer management**

Transport layer management attributes and methods are available via the TMSAP. Attributes, methods, and alerts of the TLMO are defined in 11.6.

#### **6.2.8.2.5 Security management**

Device security management attributes and methods are available via the DMSAP and TMSAP. Attributes, methods and alerts of the DSMO are defined in 7.11.

#### **6.2.8.2.6 Application sub-layer management**

Application sub-layer management attributes and methods are available via the ASMSAP. Attributes, methods, and alerts of the ASLMO are defined in 12.19.

### **6.3 System manager**

#### **6.3.1 General**

The functions of the system manager include security management, address allocation, software updating, system performance monitoring, device management, system time services, and communication configuration including contract services, and redundancy management.

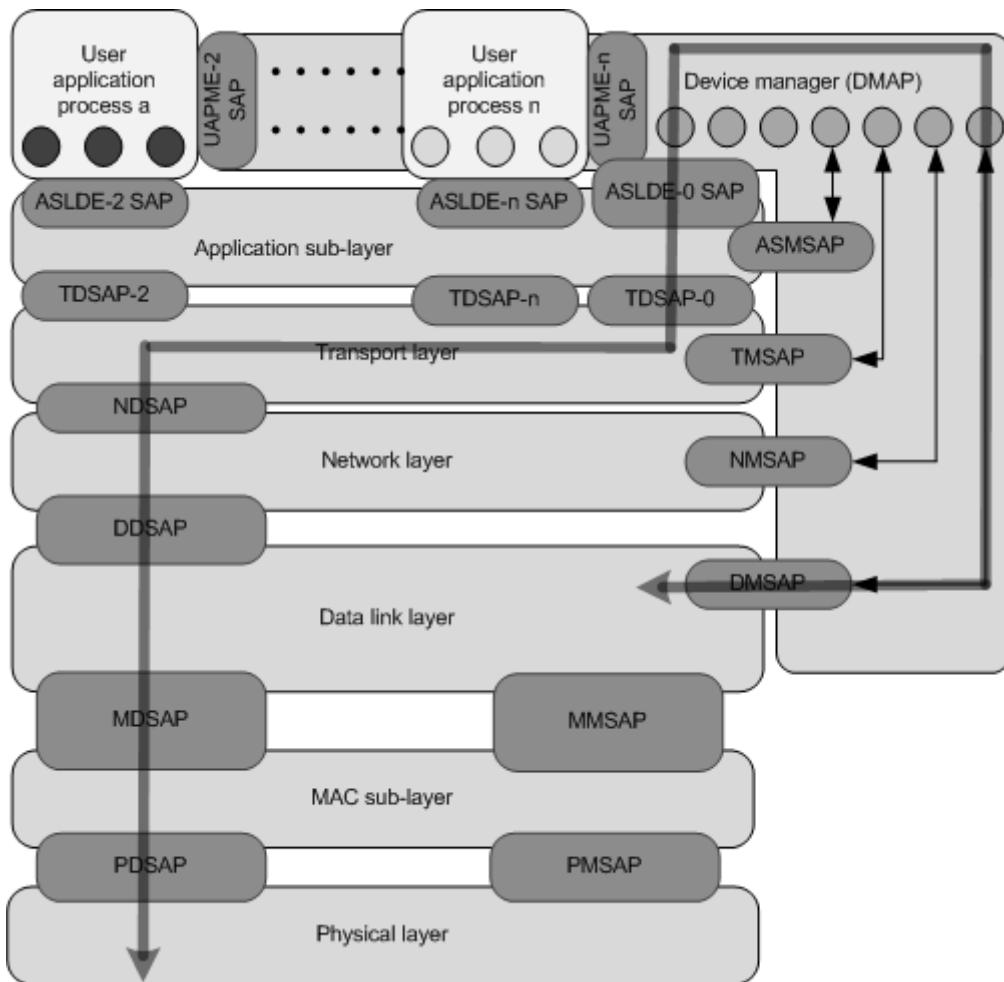
The system manager shall use ASL services to remotely access management objects in the DMAPs of devices compliant with this standard.

System manager is a role and is not tied into a specific fixed physical address.

#### **6.3.2 System management architecture**

Conceptually, the system manager can be viewed as an application process running on any device in the network. Such a device shall be capable of supporting the system manager role. The SMAP is accessible only on such a device. The SMAP shall use the application sub-layer SAP ASLDE-1 SAP for communicating with the devices. This application sub-layer SAP shall correspond to transport layer SAP TDSAP-1 which shall correspond to port number 0xF0B1.

Figure 26 shows the system manager that resides in a field device compliant with this standard.



**Figure 26 – System manager architecture concept**

As shown in Figure 26, TDSAP-1 shall be used to access the management objects in the SMAP. The definition of these system management objects is necessary to provide remote access to these functions for the devices in the network that are compliant with this standard.

### 6.3.3 Standard system management object types

Table 12 includes a list of system management object types that are specified in this standard.

**Table 12 – System management object types**

<b>Defining organization: ISA</b>			
<b>Standard object type name</b>	<b>Standard object type identifier</b>	<b>Standard object identifier</b>	<b>Object description</b>
System time service object (STSO)	100	1	This object facilitates the management of system-wide time information; see 6.3.10
Directory service object (DSO)	101	2	This object facilitates the management of addresses for all existing devices in the network; see 6.3.5
System communication configuration object (SCO)	102	3	This object facilitates the communication configuration of the system including contract establishment, modification and termination; see 6.3.11
Device management service object (DMSO)	103	4	This object facilitates device joining, device leaving, and device communication configuration; see 6.3.9
System monitoring object (SMO)	104	5	This object facilitates the monitoring of system performance; see 6.3.7
Proxy security management object (PSMO)	105	6	This object acts as a proxy for the security manager; see 6.3.4
Upload/download object (UDO)	3	7	This object facilitates downloading firmware/data to devices and uploading data from devices; see 6.3.6
Alert receiving object (ARO)	2	8	This object receives all the alerts destined for the system manager; see 6.3.7
Device provisioning service object (DPSO)	106	9	This object facilitates device provisioning; see 6.3.8
Reserved for future standard releases	107-113	—	—

Devices that require system management services communicate with the appropriate objects given above.

### **6.3.4 Security management**

The system manager interfaces with the security manager to generate keys and authenticate devices. The security manager is functionally separated from the system manager so that the security policies can be common across the networks of the administrator and other types of networks. Placing the security manager functionally behind the system manager also hides from the devices the various protocols, such as Kerberos, that may be used by a security management function. More details are provided in Clause 7.

The interface between the system manager and the security manager is not specified in this standard. Conceptually, the system manager can be viewed as including a proxy security management object (PSMO). This PSMO forwards all security related messages between the security manager and the devices in the network that are compliant with this standard. This PSMO can be used by the security manager to access information from other system management objects, such as current TAI time, if necessary. The security manager does not have a valid address as defined by this standard; thus, devices that wish to communicate with the security manager can only do so by communicating with the PSMO.

The attributes, methods and alerts of the PSMO are defined in Clause 7.

### **6.3.5 Addresses and address allocation**

#### **6.3.5.1 General**

The system manager is responsible for assigning addresses to devices when they join the network.

#### **6.3.5.2 Address types**

Every device compliant with this standard shall have one identifier and two addresses:

- Each device compliant with this standard shall have an IEEE EUI-64 identifier that is presumed to be globally unique (vendors are expected to ensure global uniqueness of these identifiers). Failover mechanisms for the gateway, system manager and security manager roles are typically ensured through redundancy. Redundancy for the purposes of

failover may involve EUI-64 identifier duplication for the redundant entities. IEEE EUI-64 identifier duplication is outside the scope of this standard.

- Each device compliant with this standard shall be assigned one 128-bit network address by the system manager when it joins the network; this 128-bit network address shall be unique across the network.
- Each device compliant with this standard that is accessible through a DL subnet shall have a DL subnet-unique 16-bit DL address for its 128-bit network address. This 16-bit DL address shall be assigned by the system manager. The scope of any 16-bit DL address is limited to a particular DL subnet. The valid range for these 16-bit addresses is 1 to 0x7fff. Address 0 is reserved to indicate that there is no address. Addresses above 0x7fff are reserved for multicast addresses (6LoWPAN), graph IDs, and future unspecified purposes.

In this standard, 16-bit DL addressing is always used within a DL subnet, with the exception that EUI-64 identifiers are used in a limited way during the join process. The join process is described in 7.4.

#### **6.3.5.3 Address allocation**

The system manager shall allocate the 128-bit network address, as well as the DL subnet-unique 16-bit DL address, to a device when it joins the network. This is described in 7.4.

When a source device that belongs to a particular DL subnet communicates over the backbone with a destination device that belongs to a different DL subnet, the system manager shall assign local DL subnet-unique 16-bit DL addresses to both devices in each other's DL subnets. Such 16-bit local DL addresses for remote devices (i.e., devices residing in another subnet) may be established by the system manager upon a contract request. (Contracts are described in 6.3.11.2.)

When the source sends a message to the destination, the 16-bit DL address of the destination shall be used at the NL and DL to construct the NPDU and DPDU. These layers can use the directory look-up service provided by the system manager to obtain the 16-bit DL address of the destination, if not already known. This service is described in 6.3.5.4.

The backbone routers shall do the address translations between the 16-bit DL address (per subnet) and 128-bit network address for a given device. Note that the 16-bit DL DL address is used only within the DL. Once a message reaches the backbone, the full 128-bit network address shall be used. Informative examples for such scenarios are given in 10.2.7.

This standard does not specify any mechanisms for how the system manager allocates the 128-bit addresses and 16-bit DL addresses. 10.2.7 contains examples for Ethernet based routing and fieldbus based routing. The Ethernet based routing example describes the use of IPv6-based 128-bit network addresses. The fieldbus routing example describes the use of non-IPv6-based 128-bit network addresses.

Devices shall only have one valid 128-bit address. Multi-homing devices that require multiple 128-bit addresses are not covered in this standard.

Addressed entities on the backbone, such as system managers and gateways, shall also be assigned 16-bit DL addresses for use within a DL subnet. Thus, most addresses used within a DL subnet will be DL addresses.

#### **6.3.5.4 Directory service**

The directory service object (DSO) in the system manager provides the necessary attributes for looking up the address translation between the EUI-64, the 128-bit network address, and the 16-bit DL address(es) of a given device. DL subnet IDs are also maintained by the DSO, but the allocation of these subnet IDs is not specified in the standard.

The attributes of the DSO are defined in Table 13.

**Table 13 – DSO attributes**

<b>Standard object type name: Directory service object (DSO)</b>				
<b>Standard object type identifier: 101</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Address_Translation_Table	1	Address translation table containing EUI-64, 128-bit network address, DL subnet ID(s) and 16-bit DL address(es) of all devices in the network	Type: Array of Address_Translation_Row	Structured attribute used to look-up address translations; see data type definition in Table 14
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
Reserved for future standard releases	2-63	—	—	—

The data structure Address\_Translation\_Row is defined in Table 14.

**Table 14 – Address\_Translation\_Row data structure**

<b>Standard data type name: Address_Translation_Row</b>		
<b>Standard data type code: 402</b>		
Element name	Element identifier	Element type
EUI64	1	Globally unique IEEE EUI-64 address of device; Type: Unsigned64 Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
Network_Address_128_Bit	2	128-bit network address of device assigned by system manager; Type: Unsigned128 Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
DL_Subnet_ID	3	DL subnet in which or from which this device is reachable; a device may be reachable from multiple subnets in which case this element corresponds to one such subnet; Type: Unsigned16 Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
DL address_16_Bit	4	16-bit DL address of device in the subnet indicated by the DL_Subnet_ID element given above; Type: Unsigned16 Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A

Address translation look-up service is provided by the Read\_Address\_Row method defined in

Table 15.

**Table 15 – Read\_Address\_Row method**

<b>Standard object type name: Directory Service object (DSO)</b>			
<b>Standard object type identifier: 101</b>			
<b>Defining organization: ISA</b>			
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>	
Read_Address_Row	1	Method to use the address translation look-up service for reading the values of other addresses / identifier of a device given an index (one of its address / identifier)	
	<b>Input arguments</b>		
	<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>
	1	Attribute_ID	Data type: Unsigned8 Value = 1 (Address_Translation_Table attribute of DSO)
	2	Index_Info	Index used for look-up; Enumeration: 0 – only EUI-64 provided 1 – only 128-bit address is provided 2 – EUI-64 and DL subnet ID are provided 3 – 128-bit address and DL subnet ID are provided 4 – DL subnet ID and 16-bit DL address are provided; See Table 16.
	3	Index_EUI64	Data type: Unsigned64 Value: EUI-64 of device for which address look-up is needed
	4	Index_128_Bit_Address	Data type: Unsigned128 Value: 128-bit network address of device for which address look-up is needed
	5	Index_DL_Subnet_ID	Data type: Unsigned16 Value: 16-bit DL subnet ID of device for which address look-up is needed
	6	Index_DL_address_16_Bit	Data type: Unsigned16 Value: 16-bit DL address of device for which address look-up is needed
	<b>Output arguments</b>		
	<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>
	1	Value_Type	Data type: Unsigned8 Indicates the type of information being provided; Enumeration: 0 – single row if DL subnet ID value provided as input argument 1 – all rows if DL subnet ID value not provided as input argument (i.e., all rows for given EUI-64 or 128-bit address are returned); See Table 17.
	2	Value_Size	Data type: Unsigned8 Number of rows being returned
	3	Data_Value_1	Data type: Address_Translation_Row EUI-64, 128-bit network address, DL subnet ID, 16 bit DL address of device
	n+3	Data_Value_n	Data type: Address_Translation_Row EUI-64, 128-bit network address, DL subnet ID, 16 bit DL address of device

Some of the input arguments are not applicable if the Index\_Info argument has certain values and so shall not be included in the request. The usage of input arguments for the Read\_Address\_Row method is described in Table 16.

**Table 16 – Input argument usage**

<b>Input argument</b>	<b>Not applicable for</b>
Attribute_ID	Index_Info value
Index_Info	—
Index_EUI64	—
Index_128_Bit_Address	1, 3, 4
Index_DL_Subnet_ID	0, 2, 4
Index_DL_Address_16_Bit	0, 1
Index_DL_Address_16_Bit	0, 1, 2, 3

Some of the output arguments are not applicable if the Value\_Type argument has certain values and so shall not be included in the response. The usage of output arguments for the Read\_Address\_Row method is described in Table 17.

**Table 17 – Output argument usage**

<b>Output argument</b>	<b>Not applicable for</b>
Value_Type	Value_Type value
Value_Size	—
Data_Value_1	0
Data_Value_n	—
Data_Value_n	0

High-side interfaces on the DSO to delete addresses or modify addresses are not specified in this standard.

### **6.3.5.5 Multicast address management**

16-bit addresses 100xxxxx xxxxxxxx shall be reserved for multicast, following the convention set by IETF RFC 4944.

Multicast address management is not specified in this standard. Additional attributes for the directory service object may be specified by vendors providing multicast address management options.

### **6.3.6 Firmware upgrading**

The system manager provides support for over-the-air firmware upgrades to devices. The system manager supports communication protocol suite firmware updates.

The system manager shall provide an interface for accepting the firmware upgrades that need to be sent to any device in the network. The system manager shall use the UDO in the DMAP of the device for sending this update to the device. Communication protocol suite firmware updates shall be performed only through the system manager UDO. This UDO is described in 12.15.2.4.

As the system manager maintains information about all the devices in the network, the host update application can obtain information about the devices that are in the network from the system manager. The host update application may use this information to determine which devices need such firmware upgrades. The gateway may also be used to send firmware upgrades to the device. If these upgrades are communication protocol suite upgrades, they must be sent through the system manager UDO. This is described in 13.3.2.

The firmware upgrade process shall assure that network operations are maintained across updates. This process may include a cut-over mechanism that specifies a cut-over time (after the update is delivered), at which point devices shall begin using the new firmware. The cut-over time shall use the shared sense of time configured by the system manager. If the system manager is sending the firmware upgrade to the device, it may send the cut-over time along with the download, or it may send the cut-over after the download is complete.

Since firmware upgrades may be vendor-specific, the updates are opaque to the system manager providing the update service. The system manager accepts updates via unspecified protocols (e.g., a user interface tool with a DVD reader) and provides updating to selected devices at specified times based on user or other input. Details of how the host update application communicates with the system manager, such as what devices should be updated, what their vendor IDs are, when the devices should be updated, and in what order they should be updated are not specified by this standard, but such functions are expected to be supported by the system manager. The system manager may schedule updates to the devices in such a manner that network downtime is either avoided or minimized. This schedule may depend on network topology.

Multicasting of firmware upgrades is not specified by this standard.

The UDO in the system manager shall be used if the firmware of the system manager itself needs to be upgraded. The attributes, methods and state machines of the UDO in the system manager are as per the definition provided in 12.15.2.4. The object identifier for the UDO in the SMAP shall be 7.

### **6.3.7 System performance monitoring**

#### **6.3.7.1 General**

System performance monitoring is done by the system manager in order to collect information that can be used to take necessary actions for optimizing system performance and for reacting to changes in the radio environment and device status. Such actions are done through system communication configuration which is described in 6.3.11.

System performance monitoring is accomplished via polling of device attributes or by configuring devices to generate alerts that provide event-driven information.

System performance monitoring using periodic publication of health reports from the devices is supported through the use of the HRCO in the DMAP of each device. HRCO is described in 6.2.7.7 and can be configured by the system manager to periodically report the values of one or more attributes in the management objects of the device. Before the system manager configures the HRCO of any particular device to publish health reports, it needs to create a unique dispersion object in the SMAP to act as the subscriber to the data that will be published by the HRCO of this particular device. Information about this unique dispersion object shall be conveyed to the HRCO of this particular device by configuring the CommunicationEndPoint attribute in the HRCO. Dispersion object is described in 12.15.2.6.

Information about the capabilities of a new device is provided to the system manager during its join process. This is discussed in 6.3.9.

Devices may be configured by the system manager to generate alerts to provide event-driven information, for example, when a link stops working or when the battery of a field device has less than 25% remaining capacity. This is described in 6.3.7.2.

While the device implementing the system manager may have an interface that allows plant operations and maintenance personnel to observe and control the performance of the network and devices, this interface is neither mandatory nor is it specified by this standard.

The UDO in the DMAP of a device may be used for downloading large blocks of device performance data from the device to the system manager. Such data may be vendor-specific, device model-specific, or device instance-specific. The UDO in the system manager may be used to upload such data for further analysis. Such data collection and analysis is not specified by this standard.

#### **6.3.7.2 System management alerts**

The system manager contains an alert receiving object (ARO) that receives communication diagnostic alerts and security alerts. Such alerts are described in 6.2.7.2. These alerts may be used by the system manager to monitor system performance and take suitable action when necessary. The object identifier for the ARO in the SMAP shall be 8.

After a device joins the network, it shall set the Alert\_Master\_Comm\_Diagnostics and Alert\_Master\_Security attributes of the ARMO to point to the system manager.

The attributes of the ARO in the system manager are as per the definition given in 12.15.2.3. The default value for the ARO.Categories attribute shall be 01100000.

The state diagram describing the handling of alert reports by the ARO is given in 12.15.2.3.

### **6.3.7.3 System monitoring object**

The attributes of the SMO are given in Table 18.

**Table 18 – Attributes of SMO in system manager**

<b>Standard object type name: System monitoring object (SMO)</b>				
<b>Standard object type identifier: 104</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Reserved for future standard releases	1-63	-	-	-

### **6.3.7.4 System monitoring configuration**

System performance monitoring may have its own dynamic configuration such that monitoring may be increased when necessary. For example, the list of active alerts may be adjusted depending on the current state of the network. For network diagnostics, if a failure mode is suspected, activating specific alerts may provide evidence of that failure. Such configuration of the system manager is not specified by this standard.

## **6.3.8 Device provisioning service**

Before a device joins the network, it requires the appropriate security credentials and network-specific information. These are provided to the device during the provisioning process. The provisioning process, the role of the system manager in this process, and the definition of the device provisioning service object (DPSO) are described in Clause 14.

## **6.3.9 Device management services**

### **6.3.9.1 General**

A device compliant with this standard may go through several phases in its operational lifetime. These phases are described in 5.2.8. The management of a device through some of these phases is performed by the system manager. Specifically, the system manager plays a role in the joining and leaving processes as well as the communication configuration of a device.

### **6.3.9.2 Join process**

#### **6.3.9.2.1 General**

A new device shall obtain the necessary provisioning information from the provisioning device. This is described in Clause 14. The Join\_Command attribute in the DMO of a device shall be used to command the device to join the network. Only the provisioning device can set the Join\_Command attribute to 1, hence explicitly triggering the join process of the device, given that the device has no connectivity with other entities in the network. The Join\_Command attribute shall be set to 1 by the provisioning device only following a successful provisioning of the device. Advertising routers shall proxy join requests at the rate indicated by the DMO attribute Proxy\_Join\_Request\_Rate (attribute 36). This rate ensures that the network is protected from denial of service attacks attempted via the proxy routers. For a description of Proxy\_Join\_Request\_Rate, see Table 10.

The system manager controls the process of new devices joining the network. Non-joined devices that implement a DL as per this standard, listen for advertisement messages from local routers whose advertisement functions are configured by the system manager. Such an advertising router shall assist during the join process of a new device by acting as a proxy for the system manager. This advertising router forwards the join request from the new device to the system manager and forwards the join response from the system manager to the new

device. A join request from the new device shall be processed by the system manager. The system manager shall generate a join response after communicating with the security manager.

The join request from a new device shall include non-security information such as the EUI-64 and capabilities of the device as well as security information of the device. The join response from the system manager shall include non-security information such as the assigned 128-bit address, assigned 16-bit DL address and contract information of the device as well as security information such as session key. The security information in the join request and join response is described in 7.4. Contracts are described in 6.3.11.2.

More details about the join process are described in 7.4.

#### **6.3.9.2.2 Device management object methods for advertising router**

The new device shall use the Proxy\_System\_Manager\_Join method and Proxy\_System\_Manager\_Contract method defined for the DMO of the advertising router to send its non-security information that is part of the join request and to get its non-security information that is part of the join response. The non-security information that is part of the join request is split into the network join request and the contract request. The non-security information that is part of the join response is split into the network join response and the contract response. Contracts are described in 6.3.11.2.

The Proxy\_System\_Manager\_Join method is defined in Table 19. The Proxy\_System\_Manager\_Contract method is defined in Table 20.

The new device shall use the methods defined in 7.4 for the DMO of the advertising router to send its security information that is part of the join request and to get its security information that is part of the join response. The use of all these methods by the new device is described in 7.4.

Access to the DMAP of the device is restricted to the SMAP present in the system manager. The joining shall be only allowed to access the Proxy\_System\_Manager\_Join and Proxy\_System\_Manager\_Contract DMO methods during the join process.

**Table 19 – Proxy\_System\_Manager\_Join method**

<b>Standard object type name: DMO (Device management object)</b>			
<b>Standard object type identifier: 127</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Proxy_System_Manager_Join	3	Method to use advertising router as proxy system manager to send network join request of a new device and get network join response	
		<b>Input arguments</b>	
	Argument #	Argument name	Argument type (data type and length)
	1	EUI64	Type: Unsigned64
	2	DL_Subnet_ID	Type: Unsigned16 DL subnet that the new device is trying to join; this is also the DL subnet of the advertising router; valid value set: 0 – 0xFFFF (0 = device is not part of any subnet)
	3	Device_Role_Capability	Type: Unsigned16 DMO attribute Device_Role_Capability; see Table 10
	4	Tag_Name	Type: VisibleString (max length: 16 octets) DMO attribute Tag_Name; see Table 10
	5	Comm_SW_Major_Version	Type: Unsigned8 DMO attribute Comm_SW_Major_Version; see Table 10
	6	Comm_SW_	Type: DMO attribute

		Minor_Version	Unsigned8	Comm_SW_Minor_Version; see Table 10
7	Software_Revision_Information	Type: VisibleString (max length: 16 octets)	DMO attribute Software_Revision_Information; see Table 10	
<b>Output arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
1	Assigned_Network_Address_128_Bit	Type: Unsigned128	This value is written to DMO attribute Network_Address_128_Bit; see Table 10	
2	Assigned_DL_Address_16_Bit	Type: Unsigned16	This value is written to DMO attribute DL_Address_16_Bit; see Table 10	
3	Assigned_Device_Role	Type: Unsigned16	This value is written to DMO attribute Assigned_Device_Role; see Table 10	
4	System_Manager_Network_Address_128_Bit	Type: Unsigned128	This value is written to DMO attribute System_Manager_128_Bit_Address; see Table 10	
5	System_Manager_DL_Address_16_Bit	Type: Unsigned16	This value is written to DMO attribute System_Manager_DL_Address_16_Bit; see Table 10	
6	System_Manager_EUI64	Type: Unsigned64	This value is written to DMO attribute System_Manager_EUI-64; see Table 10	
7	MIC	Type: Unsigned32	This value is used for protecting argument 1 thru 6 with Join key. See 7.4.4.3.2	
8	Assigned_Max_TSDU_Size	Type: Unsigned16	Indicates the maximum TSDU supported in octets which can be converted by the source into max APDU size by taking into account the TL, security, AL headers and TMIC sizes.	

**Table 20 – Proxy\_System\_Manager\_Contract method**

<b>Standard object type name: DMO (Device management object)</b>				
<b>Standard object type identifier: 127</b>				
<b>Defining organization: ISA</b>				
Method name	Method ID	<b>Method description</b>		
Proxy_System_Manager_Contract	4	Method to use advertising router as proxy system manager to send contract request of a new device and get contract response; Contracts are described in 6.3.11.2		
	<b>Input arguments</b>			
	Argument #	Argument name	Argument type (data type and length)	Argument description
	1	EUI64	Type: Unsigned64	DMO attribute EUI64; see Table 10
	<b>Output arguments</b>			
	Argument #	Argument name	Argument type (data type and length)	Argument description
	1	Contract_Response	Type: New_Device_Contract_Response (see Table 31)	Contract response to support future communication from new device to system manager; contracts are described in 6.3.11.2
	2	MIC	Type: Unsigned32	This value is used for protecting argument1 with join key. This MIC value is generated in Security Manager. Advertisement router shall not overwrite this value.

### 6.3.9.2.3 Capabilities of new device

Information about the capabilities of a new device with respect to the device role shall be provided to the system manager during the join process of the device. This information is described in Table 19.

### 6.3.9.3 Device configuration

A device is configured after joining a network. Device configuration includes obtaining communication resources to support the communication needs of the device and configuring the protocol stack of the device to use these resources to communicate. During this configuration, the system manager may take into account the capabilities of a device. A device may be reconfigured as the network changes or as the applications on the device need to change their services.

Device configuration is usually performed during the establishment of contracts. This is described in 6.3.11.2. Attributes and methods defined for the management objects of the DMAP shall be used by the system manager to configure the device.

The system manager does not configure the UAPs on the device. This is done by host applications on plant networks or by handheld maintenance tools.

### 6.3.9.4 Leave process

#### 6.3.9.4.1 General

The system manager controls the process of a previously joined device leaving the network. This leave process may be initiated by the device when it intends to leave the network, or it may be initiated by the system manager.

The leave process includes two scenarios: device restart and device reset to factory defaults.

Device restart occurs when either the device itself or the system manager cause the device to restart. Some examples that lead to this scenario are battery replacement or rebooting to apply a new firmware image. A device is reset to its factory default settings if the device is being returned to its factory default state. Some examples that lead to this scenario are the device being returned to general stock for future deployment or the device being moved to a different network.

#### 6.3.9.4.2 Device restart

A device restart process may be initiated by the device itself or by the system manager. There are two types of restarts: warm restart and restart as provisioned. In both cases, the devices will immediately initiate the join process following the restart event. An explicit writing of the Join\_Command DMO attribute to 1 is not needed following a warm restart or a restart as provisioned event.

The Join\_Command attribute in the DMO of a device shall be used to command the device to perform either a warm restart or restart as provisioned.

A device that receives the warm restart command shall reboot itself. If the Non\_Volatile\_Memory\_Capability attribute in the DMO is 1, the device shall retain the values of all constant and static attributes in all application objects present in the DMAP as well as in the UAPs of the device. All other attributes are reset to their default values. If the Non\_Volatile\_Memory\_Capability attribute in the DMO is 0, the device shall retain all the information that was provided to it during the provisioning step before it first joined the network as well as all the constant and static information present in the UAPs. All other attributes are reset to their default values and the device goes back to the restart as provisioned state.

A device that receives the restart as provisioned command shall reset all constant and static attributes in all application objects present in the DMAP regardless of the Non\_Volatile\_Memory\_Capability attribute setting present in the DMO except the DPO. A device that receives the restart as provisioned command shall reboot itself while retaining all the information that was provided to it during the provisioning step before it first joined the network. This information is described in Clause 14. This information is typically necessary for

the device to rejoin the network without having to go through the provisioning step once again. The device shall also retain all the constant and static information present in the UAPs.

Table 21 collects and presents the effects of the different join commands on various attribute sets.

**Table 21 – Effect of Different Join Commands on Attribute Sets**

Join Command Type	DMAP attributes (except DPO)	UAP Attributes	DPO Attributes
Warm restart (Join_Command =2) when Non_Volatile_Memory_Capability = 1	KEEP	KEEP	KEEP
Warm restart (Join_Command =2) when Non_Volatile_Memory_Capability = 0	CLEAR	KEEP	KEEP
Restart as provisioned (Join_Command =3)	CLEAR	KEEP	KEEP
Reset to factory defaults (Join_Command =4)	CLEAR	CLEAR	CLEAR

A firmware download may also result in a device restart. Information necessary for the device to use the new firmware and join the network may be stored in the device. The device typically goes through a restart as provisioned in such cases.

#### 6.3.9.4.3 Device reset to factory defaults

A device reset process may be initiated by the device itself or by the system manager.

The Join\_Command attribute in the DMO of a device shall be used to command the device to perform a reset. A reset command forces the device to reset to factory defaults, and all attributes are reset to their default values. The device is expected to return to the factory default state which is described in Clause 14.

#### 6.3.9.4.4 Device replacement

If an old device (i.e., joined device) is replaced by a new device (i.e., non-joined device) the system manager is expected to provide the old 128-bit network address to this replacement device. The host application or the user is expected to inform the system manager about this replacement through communication path (5) in Figure 23. The system manager may choose to configure other attributes in the DMAP of the replacement device to match those in the old device. Configuration of the UAPs in the replacement device is expected to be done by host applications on plant networks or by handheld maintenance tools.

#### 6.3.9.5 Device management service object

The device management service object (DMSO) in the system manager shall handle the non-security information in the join request from the new device that is forwarded by the advertising router and shall generate the non-security information in the join response.

The attributes of the DMSO are given in Table 22.

**Table 22 – Attributes of DMSO in system manager**

<b>Standard object type name: Device management service object (DMSO)</b>				
<b>Standard object type identifier: 103</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Reserved for future standard releases	1-63	-	-	-

A new device communicates with an advertising router which acts as a proxy for the system manager and forwards all the join messages between the new device and the system manager. The join process is described in 7.4. The methods used for sending join request and join response messages between the new device and the advertising router are given in 6.3.9.2.2. The advertising router shall use the System\_Manager\_Join method and the System\_Manager\_Contract method defined in the DMSO for sending the network join request

and the contract request and for receiving the network join response and the contract response associated with the join process of this new device.

The source object of the System\_Manager\_Join and System\_Manager\_Contract methods is the DMO of the proxy advertising router that communicates with the system manager on behalf of the new device.

The System\_Manager\_Join method is defined in Table 23. The System\_Manager\_Contract method is defined in Table 24.

**Table 23 – System\_Manager\_Join method**

<b>Standard object type name: Device management service object (DMSO)</b>			
<b>Standard object type identifier: 103</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
System_Manager_Join	1	Method to send network join request of a new device to system manager and get network join response	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	EUI64	Type: Unsigned64	DMO attribute EUI64; see Table 10
2	DL_Subnet_ID	Type: Unsigned16	DL subnet that the new device is trying to join; this is also the DL subnet of the advertising router; valid value set: 0 – 0xFFFF (0 = device is not part of any subnet)
3	Device_Role_Capability	Type: Unsigned16	DMO attribute Device_Role_Capability; see Table 10
4	Tag_Name	Type: VisibleString (max length: 16 octets)	DMO attribute Tag_Name; see Table 10
5	Comm_SW_Major_Version	Type: Unsigned8	DMO attribute Comm_SW_Major_Version; see Table 10
6	Comm_SW_Minor_Version	Type: Unsigned8	DMO attribute Comm_SW_Minor_Version; see Table 10
7	Software_Revision_Information	Type: VisibleString (max length: 16 octets)	DMO attribute Software_Revision_Information; see Table 10
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Assigned_Network_Address_128_Bit	Type: Unsigned128	This value is written to DMO attribute Network_Address_128_Bit; see Table 10
2	Assigned_DL_Address_16_Bit	Type: Unsigned16	This value is written to DMO attribute DL_Address_16_Bit; see Table 10
3	Assigned_Device_Role	Type: Unsigned16	This value is written to DMO attribute Assigned_Device_Role;

			see Table 10
4	System_Manager_Network_Address_128_Bit	Type: Unsigned128	This value is written to DMO attribute System_Manager_128_Bit_Address; see Table 10
5	System_Manager_DL_Address_16_Bit	Type: Unsigned16	This value is written to DMO attribute System_Manager_DL_Address_16_Bit; see Table 10
6	System_Manager_EUI64	Type: Unsigned64	This value is written to DMO attribute System_Manager_EUI-64; see Table 10
7	MIC	Type: Unsigned32	This value is used for protecting argument 1 thru 6. See 7.4.4.3.2
8	Assigned_Max_TSDU_Size	Type: Unsigned16	Indicates the maximum TSDU supported in octets which can be converted by the source into max APDU size by taking into account the TL, security, AL headers and TMIC sizes.

**Table 24 – System\_Manager\_Contract method**

<b>Standard object type name: Device management service object (DMSO)</b>			
<b>Standard object type identifier: 103</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
System_Manager_Contract	2	Method to send contract request of a new device to system manager and get contract response; Contracts are described in 6.3.11.2	
	<b>Input arguments</b>		
	Argument #	Argument name	Argument type (data type and length)
	1	EUI64	Type: Unsigned64 EUI-64 of the new device trying to join the network
	<b>Output arguments</b>		
	Argument #	Argument name	Argument type (data type and length)
	1	Contract_Response	Type: New_Device_Contract_Response (see Table 31) Contract response to support future communication from new device to system manager; contracts are described in 6.3.11.2
	2	MIC	Type: Unsigned32 This value is used for protecting argument 1 thru 6 with join key. This MIC value is generated in Security Manager. See 7.4.4.3.2.

When the DMSO generates the System\_Manager\_Join and System\_Manager\_Contract responses, it first sends these responses to the PSMO which in turn sends them to the security manager. The security manager shall protect these responses with MIC field using the join key and send them back to the PSMO which in turn hands them back to the DMSO. The DMSO shall then send the responses back to the advertising router. This interaction with the PSMO is described in 7.4.

### **6.3.10 System time services**

#### **6.3.10.1 General**

The time in this standard is based on International Atomic Time (TAI) as the time reference. The time in this standard is reported as elapsed seconds since the TAI instant of 00:00 on 1 January 1958 (i.e., 1958/01/01 00:00).

The system supports time synchronization so that, at the device level, applications may use time to coordinate activities or time-stamp information, improving energy use and reliability. System time shall be available from at least one device (system time source) on the network.

A TAI time source is not required for operation of a network compliant with this standard. Alternative time sources are converted to TAI units. The time base used by the network shall be within  $\pm 1$  second of actual TAI time. The gateway shall convert nominal network TAI time to the local system time reference if it is available.

The system manager shall configure at least one system time source in each DL subnet of the network. The system manager itself may be the system time source. This is described in 6.3.10.3. The system manager shall also configure the distribution topology for the dissemination of time in the DL subnet and for the synchronization of device clocks. The system manager configures each device in the DL subnet with the clock parent(s) that the device shall use for synchronizing its clock. The DL in each device is responsible for measuring time and keeping the clocks synchronized. This is described in 9.1.9.

Backbone routers are expected to use either proprietary or standardized techniques for maintaining time synchronization. These techniques are not specified by this standard.

Devices needing to convert TAI time to hh:mm:ss format, such as on a user display, may account for a Coordinated Universal Time (UTC) accumulated leap second adjustment.<sup>3</sup> The system manager shall provide this UTC adjustment to these devices. If the device needs this UTC adjustment information from the system manager, it should refresh it infrequently but periodically, such as at the start of each month or any other arbitrary clock boundary.

All devices in a network compliant with this standard share the TAI time reference with variable degrees of accuracy. To support sequence of events or other timing related operations of the application processes, all routing devices within a network compliant with this standard should be accurate to within  $\pm 10$  ms. The exact clock accuracy requirement for each routing device is described in 9.1.9.2.2.

The system manager is responsible for coordinating the time across different subnets by selecting the appropriate system time sources in each subnet. This coordination is not specified by this standard.

To support sequence of events or other timing related operations of the application processes, backbone routers also should be accurate to within  $\pm 10$  ms. Neither conversion of the time units used by the backbone routers nor adjustment to align with the TAI time being used by the devices compliant with this standard is specified by this standard.

If the system manager is part of the DL subnet, then the DL layer in the system manager is responsible for measuring time and keeping the device clock synchronized. If the system

---

<sup>3</sup> A list of such adjustments is maintained at <ftp://maia.usno.navy.mil/ser7/tai-utc.dat>.

manager is connected to the backbone, it is expected to maintain clock synchronization with the rest of the network and maintain time information in TAI units. In this case, the techniques for doing so are not specified by this standard.

Protocol layers that require the current TAI time of the device may obtain it from the DMO in the DMAP.

### **6.3.10.2 Device clock accuracy capabilities**

The system manager needs to know the clock accuracy of each device. For example, it needs to know whether a device is capable of maintaining  $\pm 1$  ms accuracy for 30 second without a clock update. Such information about a device shall be provided to the system manager by the DLMO. This is described in Table 147.

### **6.3.10.3 System time source selection**

The device implementing the system manager role may also implement the system time source role in a network, or it may delegate the system time source role to any device(s) in the network capable of playing this role as indicated by the *Device\_Role\_Capability* attribute in the DMO of the device. The system manager shall use the *Assigned\_Device\_Role* attribute in the DMO of the device to configure it as a system time source. The system manager may select the system time source based on the clock accuracy capabilities of the device.

The system time source is the ultimate source of the time sense in a DL subnet. The system time source within a DL subnet shall be accurate to within  $\pm 1$  second of actual TAI time, and shall monotonically increase at a rate that tracks TAI time with 10 ppm maximum error, i.e., the rate of increase of time shall be accurate, even if the time source itself is relatively inaccurate.

If multiple system time sources exist within a DL subnet, they should track each other within 0,1 ms. If 0,1 ms synchronization among DL subnet system time sources cannot be arranged, the system manager shall dictate when a device switches from one system time source to the other. The *dlmo.ClockStale* attribute described in 0 shall be used to inform the device when to switch over to the other system time source. Time propagation paths are described in 6.3.10.4. If devices are to switch system time sources, the time propagation paths can be re-arranged by the system manager as appropriate.

The system manager shall ensure that each DL subnet has at least one system time source. Some examples of system time sources include:

- In an outdoor application, the system manager may designate a few devices with global positioning system (GPS) capabilities as system time sources. Time may be propagated through the DL subnets from these sources.
- In a large network with an Ethernet backbone, the backbone itself may provide a time service that is synchronized to within 0,1 ms to a shared time reference for devices that are on the backbone. The system manager may designate the backbone routers as system time sources, and time may be propagated from these backbone routers to devices in the DL subnets.
- The system manager may periodically synchronize to a remote time source via a long-distance wired or wireless connection. This time source provides  $\pm 1$  second accuracy. The system manager may then act as the system time source in the network.

If multiple system time sources exist in a DL subnet, the system manager may assign one of the system time sources as the default and the others as back-ups. The techniques for such assignment are not specified by this standard.

Clock corrections within a system time source are typically applied at a rate that can ensure that the correction does not exceed 0,5 ms in a given 30 second period. Discontinuous clock corrections are supported, with devices on a DL subnet being instructed to adjust their clocks at a specific time. This is described in 9.1.9.3.6.

#### **6.3.10.4 Time distribution topology**

In addition to system time sources, the system manager also configures clock recipients and clock repeaters in a DL subnet.

All devices in a DL subnet, except for system time sources, are configured as clock recipients, i.e., they receive periodic clock updates from one or more clock sources in their immediate neighborhoods. A clock source may be a system time source or a clock repeater.

Clock repeaters are clock recipients that also act as clock sources to certain neighbors. Clock repeaters propagate time through a DL subnet. The clock accuracy requirement for a clock repeater is described in 9.1.9.2.2.

Clock source/recipient relationships, and thus time distribution topologies, are defined by the system manager. Time propagation paths in these topologies typically match routing graphs, but this is not required. Circular time propagation paths are not allowed. Clock propagation may be arranged so that clock repeaters provide updates to their recipients soon after they themselves receive their updates.

The system manager uses the DLMO of a device to configure its clock source(s). The time of a clock recipient may be updated during each interaction with a designated clock source. The selection of clock sources and the timing of clock updates are arranged by the system manager. These clock updates are described in 9.1.9.2.

#### **6.3.10.5 Monitoring of time synchronization accuracy**

System management may support mechanisms for gathering information about the accuracy of the distributed time sense, as well as for producing an alert when the sense of time between a pair of devices varies enough to cause problems within the system. The alerts described in 9.6 may be used for this purpose by the system manager.

Vendor-specified attributes in the DMO of a device may be used for gathering such information. Vendor-specified alerts from the DMO may be used for diagnosing problems related to clock synchronization and clock maintenance.

#### **6.3.10.6 System time service object**

The system manager contains the system time service object (STSO), which shall provide the UTC accumulated leap second adjustment to the devices in the network. Other vendor-specified attributes may be added to the STSO.

The attributes of the STSO in the system manager are given in Table 25.

**Table 25 – Attributes of STSO in system manager**

<b>Standard object type name: System time service object (STSO)</b>				
<b>Standard object type identifier: 100</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Current_UTC_Adjustment	1	The current value of the UTC accumulated leap second adjustment	Type: Integer16 Classification: Dynamic Accessibility: Read only Initial default value: 0 Valid value set: -32768 to +32767	Devices that need to convert TAI time to hh:mm:ss format need this adjustment from the system manager; units in seconds; note that the adjustment can be negative; note that UTC and TAI are based on different start dates but this difference is not covered by this attribute; on January 1 2009 the value changed from 33 sec to 34 sec; the mechanism used by the system manager to obtain this adjustment is not specified.
Next_UTC_Adjustment_Time	2	The TAI time when the UTC adjustment value will change from the current one	Type: TAITimeRounded Classification: Dynamic Accessibility: Read only Initial default value: See description Valid value set: -N/A	If the system manager knows the next time this UTC adjustment value will change, the SM is expected to indicate this time in TAI units; If the system manager does not know this time, it is expected to indicate the current TAI time and as a result the value of the Next_UTC_Adjustment attribute shall be same as the value of the Current_UTC_Adjustment.
Next_UTC_Adjustment	3	The next value of the UTC accumulated leap second adjustment	Type: Integer16 Classification: Dynamic Accessibility: Read only Initial default value: 34 Valid value set: -32768 to +32767	The UTC adjustment that will go into effect at the time specified by the Next_UTC_Adjustment_Time attribute.
Reserved for future standard releases	4-63	—	—	—

### 6.3.11 System communication configuration

#### 6.3.11.1 General

The system manager provides control of the runtime system communication configuration. It supports configuration of the network, including attributes of the protocol suite from DL to AL.

System communication configuration includes the assignment of slots, templates, and graphs to the devices in the network. The system manager should take into account the capabilities of the device in the network while configuring such assignments. When necessary, the system should be reconfigured to recover from failure scenarios.

#### 6.3.11.2 Contract services

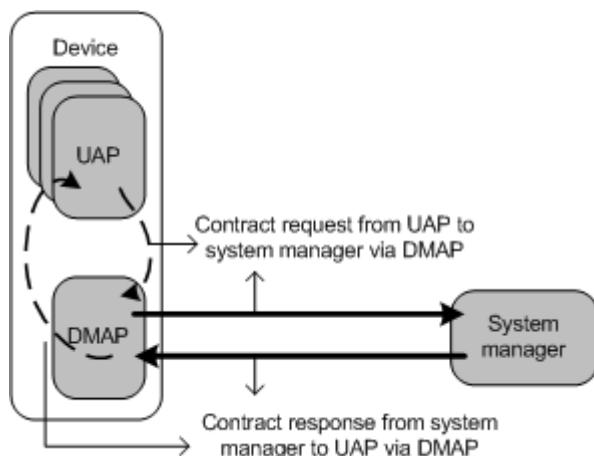
##### 6.3.11.2.1 Definition of contract

System communication configuration is achieved through the contract services provided by the system manager.

A contract refers to an agreement between the system manager and a device in the network that shall involve the allocation of network resources by the system manager to support a particular communication need of this device. This device is the source of the communication messages and the device it wants to communicate with is the destination.

A contract shall establish and support the communication path between devices in the network that are compliant with this standard to support the communication need of an application process. An application process that requires communication with an application process in another device shall request a contract. Such contract requests may originate from an application process in any one of the devices compliant with this standard, such as a field device, gateway, backbone router, or system manager.

As shown in Figure 27, contracts shall be established by the system manager. They shall also be maintained, modified and terminated by the system manager. The system manager shall interact with the affected devices in the network to perform each of these operations.



**Figure 27 – UAP-system manager interaction during contract establishment**

#### 6.3.11.2.2 Directionality of contract

Contracts shall be unidirectional, i.e., a particular contract is limited to the communication from a source to a destination. For communication in the opposite direction, a separate contract shall be established.

For a two-way communication between two devices, each device shall obtain an independent contract from the system manager in order to send its messages to the other device. The peer application processes in these devices are expected to be configured such that they request for these contracts typically before communication messages start flowing in either direction. Such configurations are done either by the system manager if the application processes are the DMAPs or by host applications on plant networks or by handheld maintenance tools if the application processes are UAPs.

#### 6.3.11.2.3 Definition of contract identifier

A contract ID (contract identifier) is a system manager-assigned identifier that shall be provided to the source after the necessary network resources have been allocated to provide the requested communication support.

The contract ID is relevant at the source, as it is used by the system manager to inform each protocol layer in the source how to treat service data units. The layers need this information before transmitting SDUs to the destination through the network. The contract requesting application process shall retrieve the assigned contract ID and shall use it to send protocol data units down the protocol suite. Protocol suite configurations at each layer for treating such upper layer protocol data units shall be referenced to the contract ID. More details are provided in 6.3.11.2.9.

Contract IDs are unique only with respect to the source, i.e., the system manager may assign the same contract ID numerical value to two independent devices to support their independent contract requests. The combination of a 128-bit source address and its contract ID shall be unique across the network.

The contract ID is also relevant at the backbone router that supports communication intended for a destination in the DL subnet supported by that backbone router. This is because the DL

in the backbone router needs to determine how to send the NPDUs through the DL subnet to its destination. Configuration of the DL in the backbone router for treating such NPDUs shall be referenced to the combination of the 128-bit source address and its contract ID. More details are provided in 6.3.11.2.9.2.

The contract ID is not relevant at any other intermediate device along the path between the source and the destination.

While contract IDs are 16-bit values, the system manager shall assign contract IDs that only fall within the range of 1 to 255 to field devices as they typically have limited memory. This way, the field devices can store contract IDs using only 8 bits. Contract ID 0 is reserved to mean no contract.

#### **6.3.11.2.4 Architecture supporting contract related messaging**

##### **6.3.11.2.4.1 General**

The DMO in each device and the system communication configuration object (SCO) in the system manager shall work together to provide contract related services such as contract establishment, contract maintenance and modification, and contract termination to each device.

##### **6.3.11.2.4.2 Handling contract-related services within device**

The DMO in each device shall be responsible for requesting, maintaining, modifying, and terminating each and every contract assigned to that device.

Any application process that requires a contract needs to send the request to the DMO which in turn shall send the request to the system manager. After the contract has been established, this application process shall not try to use network resources in excess of the ones allocated for this contract. After the contract has been established, this application process may later request for a modification or termination of this contract as appropriate.

The Contract\_Table structured attribute of the DMO may be accessed directly or indirectly by the SCO present in the system manager.

The system manager indirectly accesses the Contract\_Table structure attribute when it sends any contract related response to the device. The device shall update the Contract\_Table structured attribute of the DMO every time a contract response is received from the system manager. A new entry in the Contracts\_Table structured attribute of the DMO shall be created every time a contract response associated with the successful creation of a new contract is received from the system manager.

The system manager may directly read or write any element present in the Contract\_Table structured attribute of the DMO once the contract entry exists in the device.

##### **6.3.11.2.4.3 Handling contract related services in the network**

###### **6.3.11.2.4.3.1 General**

The SCO in the system manager is responsible for establishing, maintaining, modifying, and terminating all contracts in the network. The SCO shall coordinate with the DMO of each device to perform these operations.

###### **6.3.11.2.4.3.2 System communication configuration object**

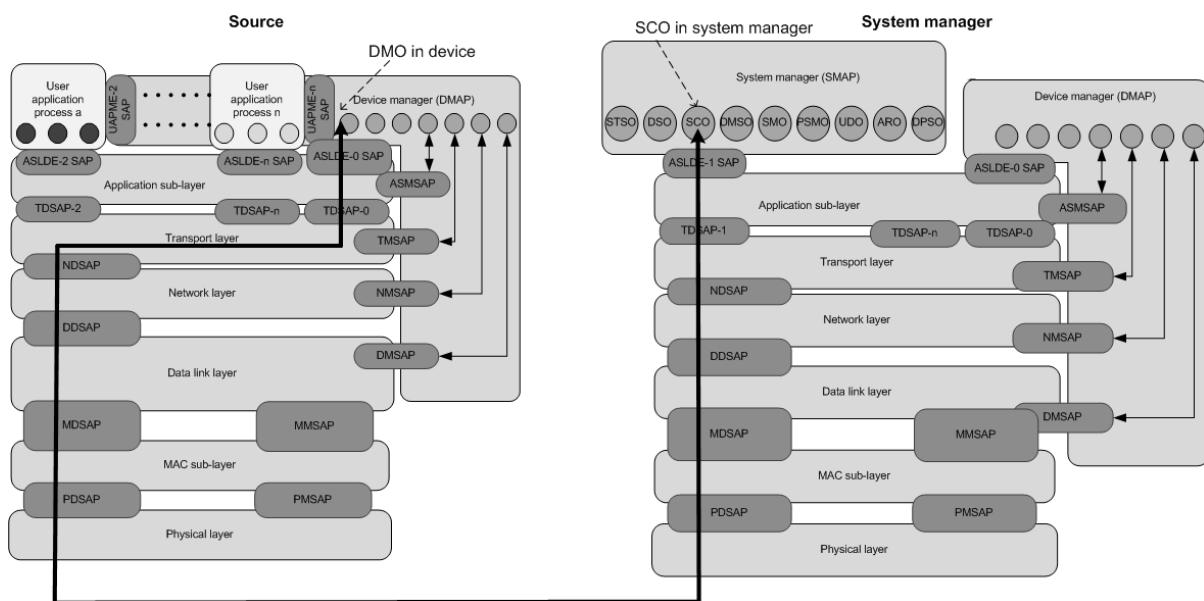
The attributes of the SCO are given in Table 26. The methods of the SCO are given in Table 27.

**Table 26 – Attributes of SCO in system manager**

<b>Standard object type name: System communication configuration object (SCO)</b>				
<b>Standard object type identifier: 102</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
Reserved for future standard releases	1-63	-	-	-

#### 6.3.11.2.4.4 Contract-related messages

All contract-related messages between the SCO in the system manager and the DMO of the device shall be application level client-server messages (i.e., writes and reads on standard object attributes and executes on standard object methods). This is illustrated in Figure 28.

**Figure 28 – Contract-related interaction between DMO and SCO**

Contract-related messages include contract requests and contract responses; these are described in 6.3.11.2.5.4.

#### 6.3.11.2.5 Contract establishment

##### 6.3.11.2.5.1 General

Contracts shall be established by the system manager when it receives a contract request. An application process, which needs to communicate with a peer process across the network, issues a contract request to the DMAP within the device. The DMO in this requesting device shall send this contract request to the SCO in the system manager. Each contract request shall include arguments that are used by the SCO to determine the network resource allocation necessary to support this request. These arguments are discussed in 6.3.11.2.5.4.

If a device receives a service request but does not already have a contract needed in order to send the service response, it should request a contract. The device shall not send the service response until the contract response is received from the SCO of the system manager and all resources needed to support the contract are successfully configured,

The algorithms used by the SCO to determine the necessary allocation of network resources are not specified in the standard, as they are all internal to the system manager. Vendors are expected to implement algorithms in the system manager that can determine the necessary allocation of network resources for the contract requests sent by the devices being managed by that system manager.

Based on this determination, the SCO shall allocate the network resources by communicating with the necessary devices in the network and providing necessary protocol suite configurations to each one of them. This shall include the configuration of the destination and the source. Details are provided in 6.3.11.2.6. Contracts shall be established for both scheduled and unscheduled communication between applications.

As part of the configuration of the source, the SCO shall also provide the contract ID. When the source receives this configuration, it shall use this contract information to start transmitting the TSDUs that needed this communication support. After the contract has been established, the source shall not try to use network resources in excess of the ones allocated for this contract.

#### **6.3.11.2.5.2 Relation between contracts and sessions**

Sessions shall be established between the TL port in the source and the corresponding TL port in the destination. All communication between these ports shall be secured using a session key that is issued by the security manager. Session establishment is described in 7.5. Contracts shall support the communication between peer application processes that reside on top of these TL ports in the protocol stack.

Multiple contracts may be established between these peer application processes to support different communication needs. As each application process in a device is associated with a TL port, all these contracts of these peer application processes shall use the same TL ports in the source and destination and so the same session key shall be used for securing all the communication that occurs using these multiple contracts.

The session key between the corresponding TL ports in the source and the destination needs to be established before a contract can be used to send messages through these TL ports. So, before the DMO sends the contract request to the SCO it is expected to check with the DSMO in the DMAP to see if a session key exists between the corresponding TL ports in the source and the destination. If such a session key does not exist, the DSMO is expected to send a session key request to the security manager and obtain a new session key. This session key request is described in 7.6.3. If a session key already exists between the corresponding TL ports, the DMO can send the contract request to the SCO immediately.

If an existing session key of a particular TL port in the device is terminated for some reason by the security manager or by the device itself, any contract that uses this particular TL port will fail as the message will not get sent down the protocol stack in the device. In this case, TL is expected to send back a failure indication to the application process that generated the message. This application process in turn is expected to send a contract request to the DMO which checks with the DSMO for a session key. The DSMO may send a new session key request to the security manager if necessary.

#### **6.3.11.2.5.3 Devices that can request contracts**

Only a traffic source shall submit a contract request to the system manager. Other devices in the network are not allowed to submit a contract request on behalf of the source.

#### **6.3.11.2.5.4 Contract request and response arguments**

Contract request arguments are pieces of information that are provided by the contract requesting application process. They are based on the communication need that the requesting application process is interested in.

**NOTE** Applications should be designed such that they request only the least amount of communication support needed to satisfy their communication needs. For example, an application that needs to publish periodic messages every 1 minute should not request a 10 second period.

The contract response arguments shall include the contract ID and other information. This information is intended to provide the basic protocol suite configuration necessary at the source.

The arguments included in the contract request and response messages are described in Table 27.

**Table 27 – SCO method for contract establishment, modification, or renewal**

<b>Standard object type name: SCO (System communication configuration object)</b>			
<b>Standard object type identifier: 102</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Contract_Establishment_Modification_Renewal	1	Method to establish a new contract / modify an existing contract / renew an existing contract by sending request to system manager	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Contract_Request_ID	Unsigned8	A numerical value, uniquely assigned by the device sending the request to the system manager, to identify the request being made. Defaults to zero, and resets to zero. Increments with each use. This ID shall be repeated if exactly the same request is re-sent due to lack of response. Rolls over to zero.
2	Request_Type	Unsigned8	Type of contract request sent to the system manager; Enumeration: 0 – new contract 1 – contract modification 2 – contract renewal Some of the input arguments below are not applicable based on this argument value; see Table 28 for details
3	Contract_ID	Unsigned16	Existing contract ID that needs to be modified or renewed
4	Communication_Service_Type	Unsigned8	Type of communication service for which the contract is being requested; Enumeration: 0 – periodic / scheduled 1 – non-periodic / unscheduled Some of the input arguments below are not applicable based on this argument value; see Table 28 for details
5	Source_SAP	Unsigned16	TDSAP in the source that will be using this contract, once it is assigned, to send application messages down the protocol stack
6	Destination_Address	Unsigned128	The address of the device that the source wants to send application messages to; note that this information may be provided to the source during provisioning or during configuration of the application process
7	Destination_SAP	Unsigned16	TDSAP in the destination that will be used to send these messages to the application layer; note that this information may be provided to the source during provisioning or during configuration of the application process
8	Contract_Negotiability	Unsigned8	Determines if the system manager can change the requested contract to meet the network resources available and if the system manager can revoke this contract to make resources available to higher priority contracts; Enumeration: 0 = negotiable and revocable 1 = negotiable but not revocable 2 = non-negotiable but revocable 3 = non-negotiable and not revocable; Contract negotiability is described in 6.3.11.2.7.2
9	Contract_Expiration_Time	Unsigned32	Determines how long the system manager should keep the contract before

<b>Standard object type name: SCO (System communication configuration object)</b> <b>Standard object type identifier: 102</b> <b>Defining organization: ISA</b>			
<b>Method name</b> <b>Method ID</b> <b>Method description</b>			
			it is terminated; units in seconds
	10	Contract_Priority	Requests a base priority for all messages sent using the contract; enumeration: 0 = best effort queued 1 = real time sequential 2 = real time buffer 3 = network control; Contract priority is described in 6.3.11.2.7.3
	11	Payload_Size	Unsigned16 Indicates the maximum payload size in octets (represented as APDU size) that the source is interested in transmitted; max value allowed is 1252 bytes
	12	Reliability_And_PublishAutoRetransmit	Unsigned8 PublishAutoRetransmit: Bit 0 indicates if source wants to retransmit old publish data if buffer is not overwritten with new publish data; bit 0 is only applicable for periodic communication and is 0 for non-periodic communication (see 12.12.2 for description); Reliability: Bits 1 to 7 indicate the requested reliability for delivering the transmitted APDUs to the destination; Bit 0 – 0 will retransmit (default), 1 will not retransmit Bits 1 to 7 – Enumeration 0 = low 1 = medium 2 = high
	13	Requested_Period	Integer16 Used for periodic communication; to identify the desired publishing period in the contract request; Valid value set: a positive number is a multiple of 1 second and a negative number is a fraction of 1 s
	14	Requested_Phase	Unsigned8 Used for periodic communication; to identify the desired phase (within the publishing period) of publications in the contract request; valid value set: 0 to 99, any other value indicates that device only cares about period and does not care about phase; see 6.3.11.2.7.4
	15	Requested_Deadline	Unsigned16 Used for periodic communication; to identify the maximum end-to-end transport delay desired; units in 10 ms
	16	Committed_Burst	Integer16 Used for non-periodic communication to identify the long term rate that needs to be supported for client-server or source-sink messages; Valid value set: positive values indicate APDUs per second, negative values indicate seconds per APDU
	17	Excess_Burst	Integer16 Used for non-periodic communication to identify the short term rate that needs to be supported for client-server or source-sink messages; Valid value set: positive values indicate APDUs per second, negative values indicate seconds per APDU
	18	Max_Send_Window_Size	Unsigned8 Used for non-periodic communication; to identify the maximum number of client requests that may be simultaneously awaiting a response
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data)	Argument description

<b>Standard object type name: SCO (System communication configuration object)</b>			
<b>Standard object type identifier: 102</b>			
<b>Defining organization: ISA</b>			
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>	
1	Contract_Request_ID	Unsigned8	The input argument Contract_Request_ID that was received in the corresponding contract request is used as this output argument
2	Response_Code	Unsigned8	Indicates if the system manager was successful or not in supporting the contract request; indicates if the source can use the contract immediately or if it has to wait; also indicates if the requested communication is being supported as is or if the system manager negotiated the request down; enumeration: 0 – success with immediate effect 1 – success with delayed effect 2 – success with immediate effect but negotiated down 3 – success with delayed effect but negotiated down 4 – failure with no further guidance 5 – failure with retry guidance 6 – failure with retry and negotiation guidance Some of the output arguments below are not applicable based on this argument value; see Table 29 for details; failure scenarios are described in 6.3.11.2.12
3	Contract_ID	Unsigned16	A numeric value uniquely assigned by the system manager to the contract being established and sent to the source. Contract IDs are unique per device. Depending on the requested resources, multiple contract request IDs from a device may be mapped to a single contract ID. In the device, the contract ID is passed in the DSAP control field of each layer and is used to look up the contracted actions that shall be taken on the associated PDU as it goes down the protocol suite at each layer (value 0 reserved to mean no contract)
4	Communication_Service_Type	Unsigned8	Type of communication service supported by this contract; Enumeration: 0 – periodic / scheduled 1 – non-periodic / unscheduled Some of the output arguments below are not applicable based on this argument value; see Table 29 for details
5	Contract_Activation_Time	TAINetworkTimeValue	Start time for the source to start using the assigned contract
6	Assigned_Contract_Expiration_Time	Unsigned32	Determines how long the system manager shall keep the contract before it is terminated; units in seconds
7	Assigned_Contract_Priority	Unsigned8	Establishes a base priority for all messages sent using the contract; enumeration: 0 = best effort queued 1 = real time sequential 2 = real time buffer 3 = network control; Contract priority is described in 6.3.11.2.7.3
8	Assigned_Max_TSDU_Size	Unsigned16	Indicates the maximum TSDU in octets which can be converted by the source into max APDU size supported by taking

<b>Standard object type name: SCO (System communication configuration object)</b> <b>Standard object type identifier: 102</b> <b>Defining organization: ISA</b>		
<b>Method name</b> <b>Method ID</b> <b>Method description</b>		
		<p>into account the TL, security, AL header and TMIC sizes;  valid value set: 70 - 1280;  The system manager shall take into account the Max_NSDU_Size constant attribute reported by the NLMOs of the source and the destination (see Table 206) while determining the value of this argument;  Fragmentation is done at the NL if the NPDU exceeds the max size of DSDU; fragmentation and reassembly is described in 10.2.5</p>
9	Assigned_Reliability_And_PublishAutoRetransmit	<p>Unsigned8</p> <p>PublishAutoRetransmit: Bit 0 indicates if retransmission of old publish data is supported if buffer is not overwritten with new publish data, bit 0 is only applicable for periodic communication and is 0 for non-periodic communication (see 12.12.2 for description);  Reliability: Bits 1 to 7 indicate the supported reliability for delivering the transmitted APDUs to the destination;  Bit 0 – 0 retransmit (default), 1 do not retransmit  Bits 1 to 7 – Enumeration  0 = low  1 = medium  2 = high</p>
10	Assigned_Period	<p>Integer16</p> <p>Used for periodic communication; to identify the assigned publishing period in the contract;  Valid value set: a positive number is a multiple of 1 second and a negative number is a fraction of 1 second.</p>
11	Assigned_Phase	<p>Unsigned8</p> <p>Used for periodic communication; to identify the assigned phase (within the publishing period) of publications in the contract; valid value set: 0 to 99</p>
12	Assigned_Deadline	<p>Unsigned16</p> <p>Used for periodic communication; to identify the maximum end-to-end transport delay supported by the assigned contract; units in 10 ms</p>
13	Assigned_Committed_Burst	<p>Integer16</p> <p>Used for non-periodic communication to identify the long term rate that is supported for client-server or source-sink messages;  Valid value set: positive values indicate APDUs per second, negative values indicate seconds per APDU.</p>
14	Assigned_Excess_Burst	<p>Integer16</p> <p>Used for non-periodic communication to identify the short term rate that is supported for client-server or source-sink messages;  Valid value set: positive values indicate APDUs per second, negative values indicate seconds per APDU.</p>
15	Assigned_Max_Send_Window_Size	<p>Unsigned8</p> <p>Used for non-periodic communication; to identify the allowed maximum number of client requests that can simultaneously await a response.</p>
16	Retry_Backoff_Time	<p>Unsigned16</p> <p>Used in the case of response code = failure with retry guidance or failure with retry and negotiation guidance; Indicates the amount of time the source should back off before resending the contract request; units in seconds; failure</p>

<b>Standard object type name: SCO (System communication configuration object)</b>			
<b>Standard object type identifier: 102</b>			
<b>Defining organization: ISA</b>			
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>	
	17	Negotiation_Guidance	Unsigned8 Used in the case of response code = failure with retry and negotiation guidance; indicates the Contract_Negotiability value supportable by system manager: 0 = negotiable and revocable 1 = negotiable but not revocable 2 = non-negotiable but revocable 3 = non-negotiable and non-revocable; Failure scenarios are described in 6.3.11.2.12
	18	Supportable_Contract_Priority	Unsigned8 Indicates the base priority supportable by system manager for all messages sent using the contract; Enumeration: 0 = best effort queued 1 = real time sequential 2 = real time buffer 3 = network control
	19	Supportable_max_TSDU_Size	Unsigned16 Indicates the maximum NSDU supportable by the system manager; units in octets (valid value set: 70 - 1280)
	20	Supportable_Reliability_And_PublishAutoRetransmit	Unsigned8 PublishAutoRetransmit: Bit 0 indicates if retransmission of old publish data is supportable if buffer is not overwritten with new publish data, bit 0 is only applicable for periodic communication and is 0 for non-periodic communication (see 12.12.2 for description) Reliability: Bits 1 to 7 indicate the reliability supportable for delivering the transmitted APDUs to the destination; Bit 0 – 0 retransmit (default), 1 do not retransmit Bits 1 to 7 – Enumeration 0 = low 1 = medium 2 = high
	21	Supportable_Period	Integer16 Used for periodic communication; to identify the supportable publishing period by the system manager; Valid value set: a positive number is a multiple of 1 s and a negative number is a fraction of 1 s
	22	Supportable_Phase	Unsigned8 Used for periodic communication; to identify the phase (within the publishing period) of publications supportable by the system manager; valid value set: 0 to 99.
	23	Supportable_Deadline	Unsigned16 Used for periodic communication; to identify the maximum end-to-end transport delay supportable by the system manager; units in 10 ms
	24	Supportable_Committed_Burst	Integer16 Used for non-periodic communication to identify the long term rate that can be supported for client-server or source-sink messages; Valid value set: positive values indicate APDUs per second, negative values indicate seconds per APDU.
	25	Supportable_Excess_Burst	Integer16 Used for non-periodic communication to identify the short term rate that can be supported for client-server or source-sink messages; Valid value set: positive values indicate APDUs per second, negative values indicate seconds per APDU.

<b>Standard object type name: SCO (System communication configuration object)</b>			
<b>Standard object type identifier: 102</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
	26	Supportable_Max_Send_Window_Size	Unsigned8 Used for non-periodic communication; to identify the supportable maximum number of client requests that can simultaneously await a response.

Table 27 also contains input and output arguments that shall be used for contract modification and contract renewal. Contract modification and contract renewal are discussed in 6.3.11.2.11.

Table 27 also contains output arguments that shall be used for failure scenarios when the system manager is not able to support the contract request. These failure scenarios are discussed in 6.3.11.2.12.

Some of the input arguments in Table 27 are not applicable when the Request\_Type and/or Communication\_Service\_Type arguments are given certain values and so shall not be included in the request. This information is provided in Table 28.

**Table 28 – Input argument usage**

Input argument	Not applicable for	
	Request_Type value	Communication_Service_Type value
Contract_Request_ID	—	—
Request_Type	—	—
Contract_ID	0	—
Communication_Service_Type	—	—
Source_SAP	—	—
Destination_Address	—	—
Destination_SAP	—	—
Contract_Negotiability	—	—
Contract_Expiration_Time	—	—
Contract_Priority	—	—
Payload_Size	—	—
Reliability_And_PublishAutoRetransmit	—	—
Requested_Period	—	1
Requested_Phase	—	1
Requested_Deadline	—	1
Committed_Burst	—	0
Excess_Burst	—	0
Max_Send_Window_Size	—	0

Some of the output arguments in Table 27 are not applicable when the Response\_Code and/or Communication\_Service\_Type arguments are given certain values and so shall not be included in the response. This information is provided in Table 29.

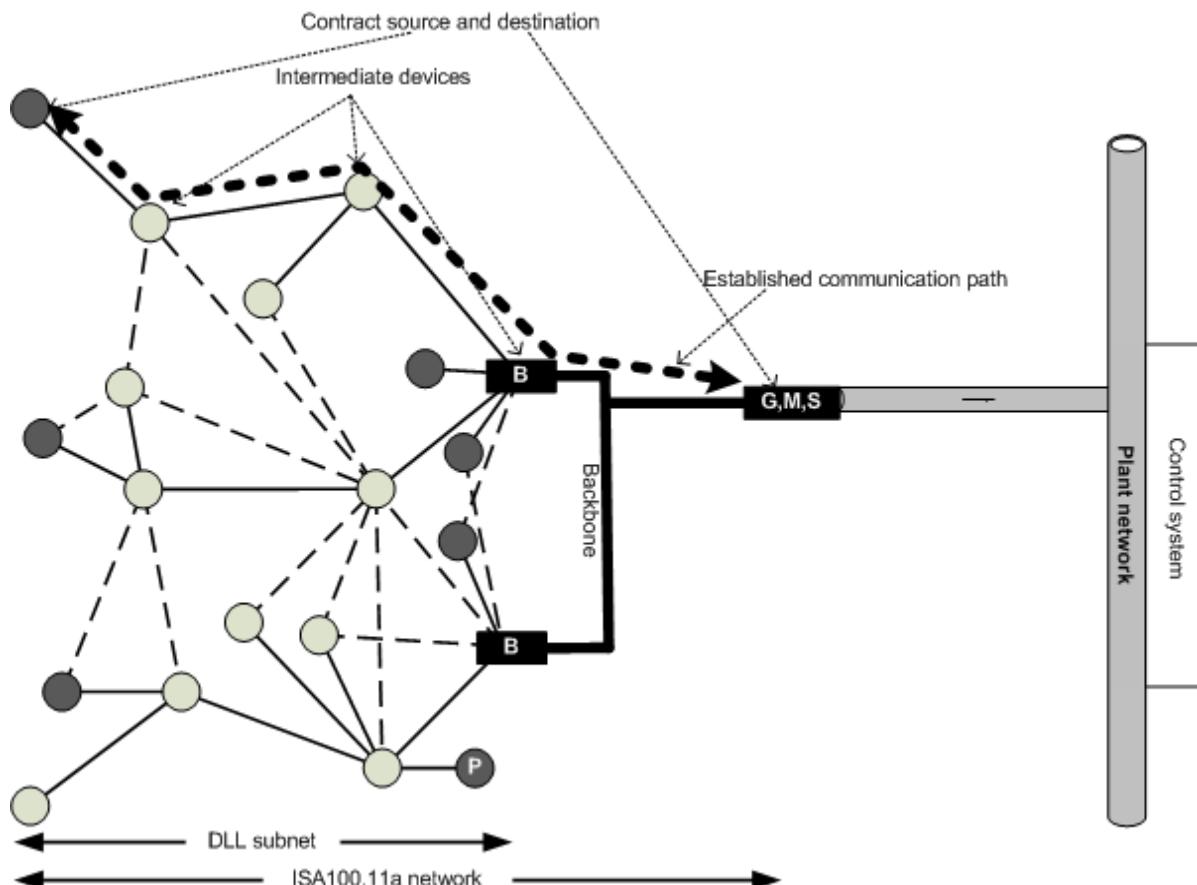
**Table 29 – Output argument usage**

<b>Output argument</b>	<b>Not applicable for</b>	
	<b>Response_Code value</b>	<b>Communication_Service_Type value</b>
Contract_Request_ID	—	—
Response_Code	—	—
Contract_ID	4, 5, 6	—
Communication_Service_Type	4, 5	—
Contract_Activation_Time	0, 2, 4, 5, 6	—
Assigned_Contract_Expiration_Time	4, 5, 6	—
Assigned_Contract_Priority	4, 5, 6	—
Assigned_Max_TSDU_Size	4, 5, 6	—
Assigned_Reliability_And_PublishAutoRetransmit	4, 5, 6	—
Assigned_Period	4, 5, 6	1
Assigned_Phase	4, 5, 6	1
Assigned_Deadline	4, 5, 6	1
Assigned_Committed_Burst	4, 5, 6	0
Assigned_Excess_Burst	4, 5, 6	0
Assigned_Max_Send_Window_Size	4, 5, 6	0
Retry_Backoff_Time	0, 1, 2, 3, 4	—
Negotiation_Guidance	0, 1, 2, 3, 4, 5	—
Supportable_Contract_Priority	0, 1, 2, 3, 4, 5	—
Supportable_max_TSDU_Size	0, 1, 2, 3, 4, 5	—
Supportable_Reliability_And_PublishAutoRetransmit	0, 1, 2, 3, 4, 5	—
Supportable_Period	0, 1, 2, 3, 4, 5	1
Supportable_Phase	0, 1, 2, 3, 4, 5	1
Supportable_Deadline	0, 1, 2, 3, 4, 5	1
Supportable_Committed_Burst	0, 1, 2, 3, 4, 5	0
Supportable_Excess_Burst	0, 1, 2, 3, 4, 5	0
Supportable_Max_Send_Window_Size	0, 1, 2, 3, 4, 5	0

### 6.3.11.2.6 Protocol suite configuration

#### 6.3.11.2.6.1 General

As part of contract establishment, the SCO shall configure the necessary devices in the network by providing necessary protocol suite configurations to each one of them. This shall include the configuration of the destination and the source, as illustrated in Figure 29.



**Figure 29 – Contract source, destination, and intermediate devices**

Intermediate devices in the network that support the communication path being established between the source and the destination shall be configured by the SCO. Such intermediate devices along the path may include both field routers and backbone routers.

#### 6.3.11.2.6.2 Configuration of intermediate field routers

Configuration of intermediate field routers shall be limited to the DL in each field router, as the message from the source to the destination traverses only through the DL of each field router along the path.

Attributes and methods defined for the DLMO of the field routers shall be used by the system manager to configure the intermediate field routers.

#### 6.3.11.2.6.3 Configuration of intermediate backbone routers

Configuration of intermediate backbone routers shall be limited to the network layer and, in some cases, the DL in each backbone router, as the message from the source to the destination traverses through the DL in the case of backbone routers that belong to the corresponding source and destination DL subnets, and the network layer of the backbone routers along the path.

Attributes and methods defined for the DLMO and NLMO of the backbone routers shall be used by the system manager to configure the intermediate backbone routers.

#### 6.3.11.2.6.4 Configuration of destination

Configuration of destination shall include the configuration of all the protocol layers. The attributes and methods defined for the DLMO, NLMO, and TLMO shall be used by the system manager to configure the destination.

### **6.3.11.2.6.5 Configuration of source**

#### **6.3.11.2.6.5.1 General**

The output arguments described in Table 27 are used at various layers of the source to determine the treatment of PDUs belonging to this contract.

The attributes and methods defined for the DLMO, NLMO, and TLMO shall be used by the system manager to configure the source.

A contract response shall be sent to the source either after all necessary network resources have been configured or after the system manager determines the time it would take to configure all necessary network resources. Depending on the situation, the system manager shall indicate if the assigned contract can be used with immediate effect or with delayed effect. The message sequence diagram in Figure 30 illustrates the case of immediate effect.

After the contract has been established, the source shall not try to use network resources in excess of the ones allocated for this contract.

#### **6.3.11.2.6.5.2 Contract information in device management object**

The DMO in the source shall maintain a list of all assigned contracts using the Contracts\_Table attribute. This attribute shall be based on the data structure Contract\_Data. When a new contract gets established, a new row shall be added to this Contracts\_Table attribute with the relevant contract information. When an existing contract gets modified or terminated, the corresponding row shall be modified or deleted in this Contracts\_Table attribute.

The SCO can also modify the parameters of the Contract\_Table attributes by accessing them directly without exchanging entire Contract\_Data structures.

The elements of the data structure Contract\_Data are defined in Table 30.

**Table 30 – Contract\_Data data structure**

<b>Standard data type name: Contract_Data</b>		
<b>Standard data type code: 401</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
Contract_ID*	1	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: 1 – 0xFFFF (0 reserved to mean no contract); This element is the same as output argument Contract_ID in Table 27. * This element is used as the index field for methods described in Table 33 and Table 34
Contract_Status	2	Type: Unsigned8 Classification: Static Accessibility: Read only Default value = N/A Valid value set: Enumeration representing: 0 – success with immediate effect 1 – success with delayed effect 2 – success with immediate effect but negotiated down 3 – success with delayed effect but negotiated down; This element is related to the output argument Response_Code in Table 27
Communication_Service_Type	3	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: Enumeration: 0 – periodic / scheduled communication 1 – non-periodic / unscheduled communication; This element is the same as output argument Communication_Service_Type in Table 27
Contract_Activation_Time	4	Type: TAI Network Time Value Classification: Static Accessibility: Read/write Default value: N/A Valid value set: See 12.22.4.2 This element is the same as output argument Contract_Activation_Time in Table 27
Source_SAP	5	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: N/A; This element is the same as input argument Source_SAP in Table 27
Destination_Address	6	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: N/A; This element is the same as input argument Destination_Address in Table 27
Destination_SAP	7	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: N/A; This element is the same as input argument Destination_SAP in Table 27
Assigned_Contract_Expiration_Time	8	Type: Unsigned32 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 0 – 0xFFFFFFFF (units in seconds);

<b>Standard data type name: Contract_Data</b>		
<b>Standard data type code: 401</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
		This element is the same as output argument Assigned_Contract_Expiration_Time in Table 27
Assigned_Contract_Priority	9	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: Enumeration: 0 = best effort queued 1 = real time sequential 2 = real time buffer 3 = network control; This element is the same as output argument Assigned_Contract_Priority in Table 27
Assigned_Max_TSDU_Size	10	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 70 – 1280; This element is the same as output argument Assigned_Max_TSDU_Size in Table 27
Assigned_Reliability_And_PublishAutoRetransmit	11	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: Bit 0 – 0 Bits 1 to 7 – Enumeration: 0 = low 1 = medium 2 = high; This element is the same as output argument Assigned_Reliability_And_PublishAutoRetransmit in Table 27
Assigned_Period	12	Type: Integer16 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: a positive number is a multiple of 1 second and a negative number is a fraction of 1 second; This element is the same as output argument Assigned_Period in Table 27
Assigned_Phase	13	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 0 to 99; This element is the same as output argument Assigned_Phase in Table 27
Assigned_Deadline	14	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 0 – 0xFFFF (units in 10 ms); This element is the same as output argument Assigned_Deadline in Table 27
Assigned_Committed_Burst	15	Type: Integer16 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 0 – 0xFFFF (units in APDUs per second for positive values, units in seconds per APDU for negative values); This element is the same as output argument Assigned_Committed_Burst in Table 27
Assigned_Excess_Burst	16	Type: Integer16

Standard data type name: Contract_Data		
Standard data type code: 401		
Element name	Element identifier	Element type
		Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 0 – 0xFFFF (units in APDUs per second for positive values, units in seconds per APDU for negative values); This element is the same as output argument Assigned_Excess_Burst in Table 27
Assigned_Max_Send_Window_Size	17	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 0 – 0xFF; This element is the same as output argument Assigned_Max_Send_Window_Size in Table 27

### 6.3.11.2.6.6 Configuration of new device

The process for a new device to join is described in 7.4. As part of the join process for a new device, a contract between the new device and the system manager shall be established.

The new device shall use the Proxy\_System\_Manager\_Contract method defined for the DMO of the advertising router to send this contract request, which is then forwarded to the system manager, and to get the contract response from the system manager via the advertising router. The Proxy\_System\_Manager\_Contract method is defined in Table 20. The advertising router shall use the System\_Manager\_Contract method defined in the DMSO for forwarding this contract request and for receiving the contract response associated with the join process of this new device. The System\_Manager\_Contract method is defined in Table 24. The DMSO works with the SCO to generate this contract response. When the new device gets this contract response, a new row shall be added to the Contracts\_Table attribute in the DMO of the new device with the relevant contract information.

The output arguments in both these methods shall be based on the data structure New\_Device\_Contract\_Response. The elements of the data structure New\_Device\_Contract\_Response are defined in Table 31.

**Table 31 – New\_Device\_Contract\_Response data structure**

<b>Standard data type name: New_Device_Contract_Response</b>		
<b>Standard data type code: 405</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
Contract_ID	1	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: 1 – 0xFFFF (0 reserved to mean no contract); This element is related to the output argument Contract_ID in Table 27
Assigned_Max_TSDU_Size	2	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 70 – 1280; This element is related to the output argument Assigned_Max_TSDU_Size in Table 27
Assigned_Committed_Burst	3	Type: Integer16 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 0 – 0xFFFF (units in APDUs per second for positive values, units in seconds per APDU for negative values); This element is related to the output argument Assigned_Committed_Burst in Table 27
Assigned_Excess_Burst	4	Type: Integer16 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 0 – 0xFFFF (units in APDUs per second for positive values, units in seconds per APDU for negative values); This element is related to the output argument Assigned_Excess_Burst in Table 27
Assigned_Max_Send_Window_Size	5	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value: N/A Valid value set: 0 – 0xFF; This element is related to the output argument Assigned_Max_Send_Window_Size in Table 27
NL_Header_Include_Contract_Flag	6	Type: Boolean Classification: Static Accessibility: Read/write Default value : N/A Valid value set: 0 = do not include, 1 = include; This element is related to the corresponding element in Table 208 and is used for configuring the NL of the new device to use the contract assigned to the new device.
NL_Next_Hop	7	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: 0 – 2 <sup>128</sup> -1; This element is related to the corresponding element in Table 208 and is used for configuring the NL of the new device to use the contract assigned to the new device.
NL_NWK_HopLimit	8	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: 0 – 255; This element is related to the corresponding element in Table 208 and is used for configuring the NL of the new device to use the contract assigned to the new device.
NL_Outgoing_Interface	9	Type: Boolean Classification: Static Accessibility: Read/write

<b>Standard data type name: New_Device_Contract_Response</b>		
<b>Standard data type code: 405</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
		Default value : N/A Valid value set: 0-DL, 1-Backbone; This element is related to the corresponding element in Table 208 and is used for configuring the NL of the new device to use the contract assigned to the new device.

The new device shall use the DL information provided in the advertisement to support this contract. After the device joins the network, the system manager shall access the relevant DLMO attributes in the device to modify this DL information as appropriate. More information about this DL information and the DLMO attributes is given in 9.1.14.

If the new device is not allowed by the security manager to join the network, the security manager should inform the system manager to free up this contract and the associated network resources. The system manager shall free up the contract and the associated network resources of such a device that is not allowed to join the network.

### **6.3.11.2.7 Quality of service**

#### **6.3.11.2.7.1 General**

The contract assigned by the system manager to a requesting application process also indicates the quality of service (QoS) for the provided communication service. The contract establishment shall be used to reach this QoS agreement between the requesting application process and the system manager.

An application process that wants to communicate with its peer may indicate the QoS desired for this communication in its contract request. The input arguments described in Table 27 may be used for this purpose.

The input arguments Contract\_Priority and Payload\_Size may be used in contract requests pertaining to both periodic and non-periodic communication services. Input arguments Requested\_Period, Requested\_Phase and Requested\_Deadline are relevant for periodic communications. Input arguments Committed\_Burst, Excess\_Burst and Max\_Send\_Window\_Size are relevant for non-periodic communications. The input argument Reliability\_And\_PublishAutoRetransmit contains information about desired reliability which is relevant for both periodic as well as non-periodic communications. It also indicates if the application process wants to retransmit old periodic communication data if new data is not available.

In the contract response, the system manager shall indicate the QoS level provided for the assigned communication service. The output arguments described in Table 27 corresponding to the above mentioned input arguments shall be used for this purpose.

#### **6.3.11.2.7.2 Contract negotiability**

A source that is sending a contract request shall also indicate whether the requested communication service and the QoS is negotiable, i.e., whether the system manager can assign a contract that provides a different communication service and QoS than the ones requested if it cannot support the request as is, and whether the system manager can revoke the contract if necessary. The input argument Contract\_Negotiability shall be used for this purpose.

Table 27 contains arguments that are necessary for contract negotiation between the source and the system manager. If the system manager is unable to support a contract request, it may choose to provide contract negotiation guidance. Such guidance shall be provided using the output arguments in Table 27 that start with the word supportable, e.g., Supportable\_Contract\_Priority.

If the system manager is unable to support the contract request at the time it was received but it expects to be able to support such a request in the future, it may indicate this by using the output argument `Retry_Backoff_Time`.

#### **6.3.11.2.7.3 Contract priorities and message priorities**

Two priority levels shall be supported in the system, contract priority and message priority.

Contract priority shall establish a base priority for all messages sent using that contract. Four contract priorities shall be supported using 2 bits:

- Network control = 3. Network control may be used for critical management of the network by the system manager.
- Real time buffer = 2. Real time buffer may be used for periodic communications in which the message buffer is overwritten whenever a newer message is generated.
- Real time sequential = 1. Real time sequential may be used for applications such as voice or video that need sequential delivery of messages.
- Best effort queued = 0. Best effort queued may be used for client-server communications.

Message priority shall establish priority within a contract. Two message priorities shall be supported using 1 bit, low = 0 and high = 1. Another 1 bit is reserved for future releases of this standard and shall be set to 0.

Contract priority shall be specified by the application, during contract establishment time, in its contract request. It may be used by the system manager to establish preferred routes for high priority contracts and for load balancing the network. The system manager shall convey the assigned contract priority to the source in the contract response.

Message priority shall be supplied by the application for every message sent down the protocol suite. In the source, the message priority shall flow down the protocol suite. The contract priority shall be added at the network layer. Contract priority shall have precedence over message priority.

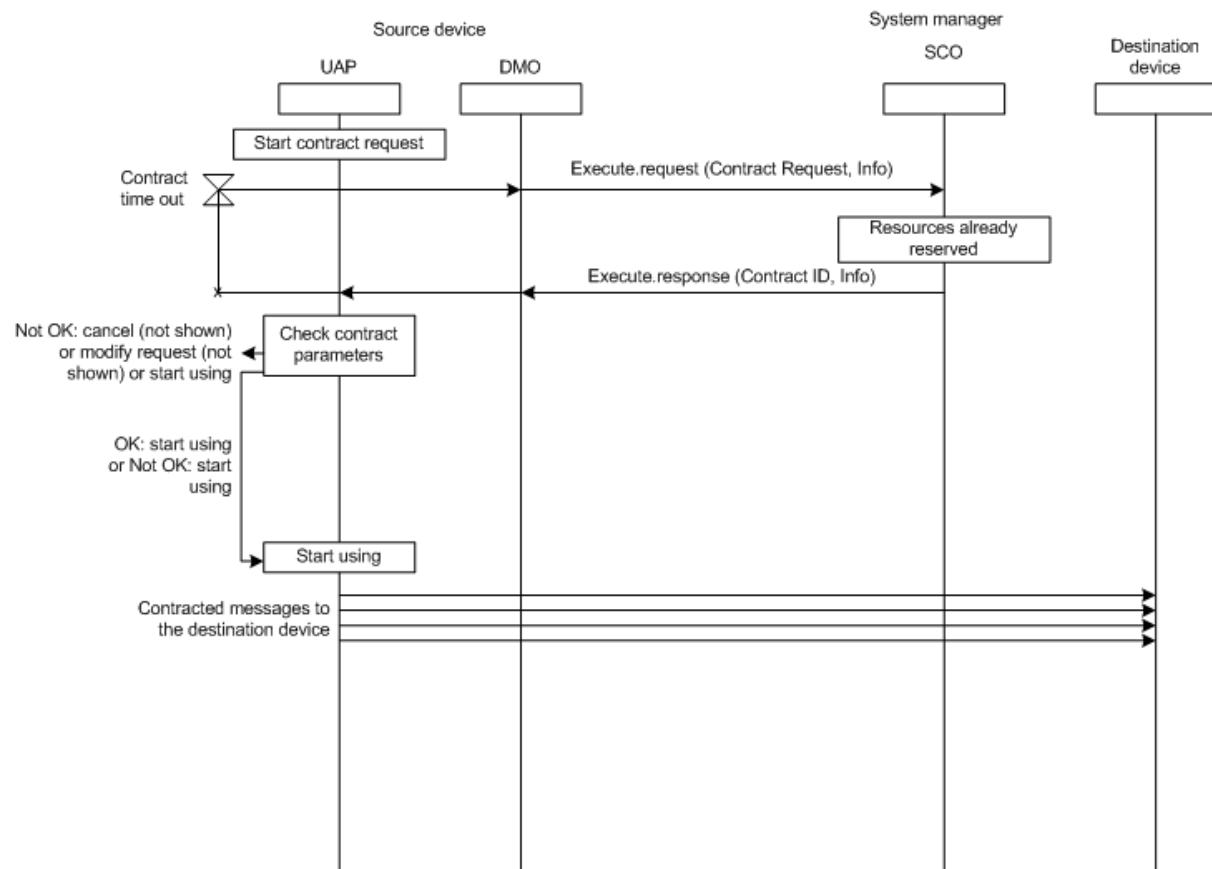
Combined contract/message priority shall be used to resolve contention for scarce resources when these messages are forwarded through the network. DL shall use this information to drive queuing decisions when forwarding messages on the DL subnet. It shall be included only in the DL header. When message is sent on a backbone, priority shall be included in the network headers. The network layer shall use priority to drive queuing decisions on a backbone.

#### **6.3.11.2.7.4 Arguments related to phase**

The input argument `Requested_Phase` shall be used by the application process requesting the contract to request a phase which is the time offset from the beginning of a period. This time offset is expressed as a percentage of the time within a period. All periods shall be calculated such that their start times are always set to the beginning of TAI time. This argument is typically used by applications that use such phase settings to achieve time synchronized distributed loop execution with minimum latency and bounded jitter. The exact timing of the phase as it relates to the DL is specified by the link number which is described in 9.4.3.7.

#### **6.3.11.2.8 Contract establishment message sequence diagram**

Figure 30 shows an example of a message sequence chart for the establishment of a contract. This example does not involve any time-outs and the source device accepts the contract established by the system manager even if this contract provides a different communication service than the one requested.



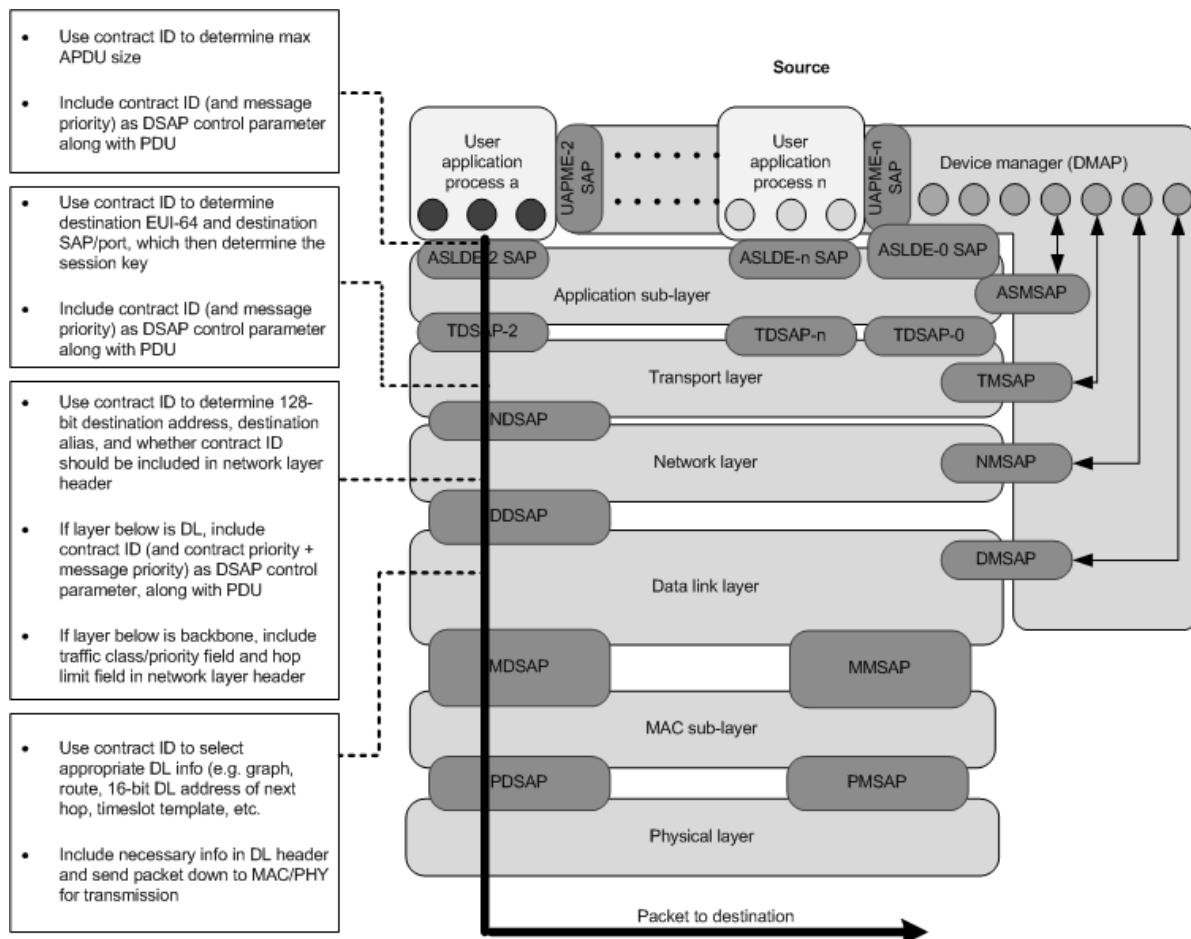
**Figure 30 – Contract establishment example**

#### 6.3.11.2.9 Use of contract identifier

##### 6.3.11.2.9.1 General

The contract ID shall be provided by the system manager to the source. The contract requesting application process shall retrieve the assigned contract ID and shall use it to send protocol data units down the protocol suite. As described previously, each layer of the source is configured for treating such upper layer protocol data units that are accompanied by the contract ID, which is passed along as a DSAP control parameter.

Figure 31 illustrates how the contract ID shall be used as the data unit flows down the protocol suite of the source.

**Figure 31 – Contract ID usage in source**

#### 6.3.11.2.9.2 Use of contract identifier in intermediate backbone routers

Inclusion of the contract ID in the network header of the NPDU by the source shall be configured by the system manager. If the communication path from the source to the destination goes through the backbone, then the system manager shall inform the source to include the contract ID in its network header. More details are provided in 10.5.3.

#### 6.3.11.2.9.3 Relation between contracts and alerts

Access to the DMAP is restricted to the SMAP that resides in the system manager. In contradiction to this general principle, alert masters are allowed to access the ARMO object present in the DMAP. DMAP access by alert masters shall be limited to the ARMO, unless the alert master uses the DMAP-SMAP session established when the device joined the network. The ARMO in the DMAP shall transmit alerts that belong to the different alert categories to the respective alert masters which are described in Table 7. If these alert masters are different devices with their own unique network addresses, the ARMO shall have a separate contract with each one to communicate the alerts. The ARMO in the device requests for these contracts from the system manager through the DMO in the device.

#### 6.3.11.2.10 Contract termination, deactivation and reactivation

##### 6.3.11.2.10.1 General

Contracts may be terminated when the communication need that established the contract has been satisfied. Contracts may also be terminated when either the source or the destination are no longer available.

When there is a contract termination, the SCO shall inform the DMO of the source, if the source is still available. The DMO in turn informs the application process that was using this contract.

When there is a contract termination, the SCO may also free up the network resources that were allocated for supporting the contract. In addition, security information, including session keys between the source and the destination, may also be deleted by the security manager based on interactions with the system manager.

A contract may also be deactivated if the communication need is expected to be suspended for a period of time. The contract can be reactivated when the communication need resumes.

#### **6.3.11.2.10.2 Contract termination when a device leaves the network or is no longer available**

When the system manager determines that a device is no longer part of the network, it shall terminate all the contracts associated with that device and free up the network resources that were allocated for supporting those contracts. The system manager may use information from other devices in the neighborhood of this device to decide that this device is no longer part of the network. The system manager may read the dlimo.Neighbor attribute (described in 9.4.3.4) of these neighboring devices to make this decision.

When a device that has DMO attribute Non\_Volatile\_Memory\_Capability = 1 loses network connectivity / power cycles / goes through a warm restart for any reason, it shall maintain all necessary information related to contracts as described in 6.3.9.4.2. So, this device can resume normal operation as soon as it re-establishes time synchronization with the network. The device is expected to re-establish time synchronization by listening for advertisements or by soliciting advertisements.

If the system manager had terminated all the contracts of this device while the device was not part of the network, the device is expected to be unsuccessful in resuming normal operation and so is expected to restart as provisioned. This device shall retain all the information that was provided to it during the provisioning step before it first joined the network as well as all the constant and static information present in the UAPs.

When a device that has DMO attribute Non\_Volatile\_Memory\_Capability = 0 loses network connectivity / power cycles / restarts as provisioned for any reason, it is expected to go through the join process again by using the information that was provided to it during the provisioning step before it first joined the network. This device shall also retain all the constant and static information present in the UAPs.

The DMO of a device that is resetting to the factory default state or is restarting as provisioned shall terminate all its contracts by using the method defined in Table 27 before resetting or restarting.

#### **6.3.11.2.10.3 Contract termination when the session key is terminated**

The system manager may terminate a contract of a particular device if it is informed by the security manager that a corresponding session key of that device has been terminated. Any contract of the device that uses the TL port corresponding to this particular session key may be terminated.

#### **6.3.11.2.10.4 Devices that can terminate, deactivate and reactivate contracts**

Only the source or the system manager shall have the ability to terminate an existing contract of the source.

Only the source shall have the ability to deactivate and reactivate an existing contract of the source.

#### **6.3.11.2.10.5 Contract termination, deactivation, and reactivation request and response arguments**

If the source decides to terminate a contract, it shall send a contract termination request to the SCO. The SCO shall then send the response back to the source informing it that the contract has been terminated. The request shall be an Execute.Request to the SCO with the contract ID as one of the input arguments, and the response shall be an Execute.Response with status as the output argument. This is described in Table 32.

If the source decides to deactivate / reactivate a contract, it shall send a contract deactivation / reactivation request to the SCO. The SCO shall then send the response back to the source informing it that the contract has been deactivated / reactivated. The request shall be an Execute.Request to the SCO with the contract ID as one of the input arguments, and the response shall be an Execute.Response with status as the output argument. This is described in Table 32.

**Table 32 – SCO method for contract termination, deactivation and reactivation**

<b>Standard object type name: SCO (System communication configuration object)</b>			
<b>Standard object type identifier: 102</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Contract_Termination _Deactivation_Reactivation	2	Method to terminate, deactivate or reactivate a contract	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Contract ID	Unsigned16	ID of contract being terminated, deactivated or reactivated
2	Operation	Unsigned8	0 – contract termination 1 – contract deactivation 2 – contract reactivation
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Status	Unsigned8	0 – success, 1 – failure

If the system manager decides to terminate a contract, it shall send a contract termination command with the contract ID to the DMO of the source. The DMO shall then return a response with the status.

The DMO method to terminate an existing contract is described in Table 33.

**Table 33 – DMO method to terminate contract**

<b>Standard object type name: Device management object (DMO)</b>		
<b>Standard object type identifier: 127</b>		
<b>Defining organization: ISA</b>		
Method name	Method ID	<b>Method description</b>
Terminate_Contract	1	Method to terminate an existing contract in the Contracts_Table attribute in Table 10. This method uses the Delete_Row method template defined in Table 521 with the following arguments: Attribute_ID: 26 (Contracts_Table) Index_1: 1 (Contract_ID)

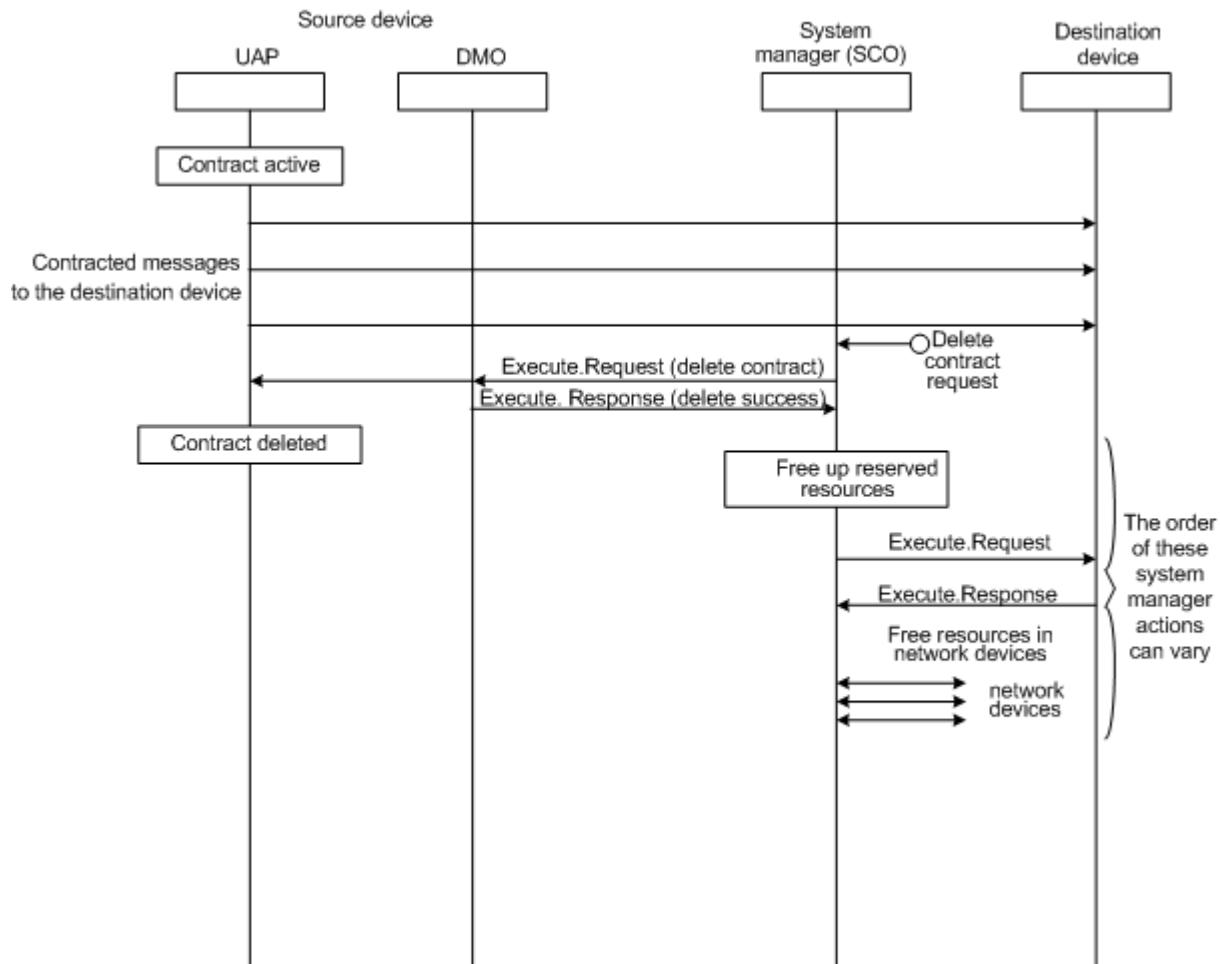
#### 6.3.11.2.10.6 Protocol suite configuration

When the SCO terminates a contract, in addition to informing the source about the termination, it may also free up the network resources that were allocated in the source, destination, and intermediate devices. Procedures similar to those used for protocol suite configuration during contract establishment (see 6.3.11.2.6) may be used by the SCO to free up these network resources.

The SCO informs the security manager through the PSMO about the contract termination. The security manager may decide to delete the session key that has been assigned for the communication between the source and the destination. In this case, the security manager shall send session key delete messages to the source and the destination through the PSMO.

### 6.3.11.2.10.7 Contract termination message sequence diagram

Figure 32 shows the message sequence chart for termination of a contract initiated by the system manager.



**Figure 32 – Contract termination**

### 6.3.11.2.11 Contract maintenance and modification

#### 6.3.11.2.11.1 General

The SCO needs to maintain established contracts by ensuring that the allocated network resources are available under normal conditions. If the allocated network resources become unavailable, the SCO may choose to allocate alternate network resources in order to continue to maintain the established contract.

A contract may be modified if the communication need of the corresponding application (supported by that contract) changes. A contract may also be modified if the system manager decides to change the network resources allocated for the contract.

Contract modifications fall into two categories:

- Modifications resulting in a reduction of the allocated network resources.
- Modifications resulting in a change or increase of allocated network resources.

For application-initiated contract modifications, these two categories follow slightly different steps.

Contract modifications that result in a reduction of the allocated network resources may go into immediate effect, i.e., the source may start using the protocol suite configuration of the modified contract as soon as it receives the response along with this configuration information from the SCO if this response indicates so in the Response\_Code output argument.

Contract modifications that result in an increase or change of the allocated network resources shall not go into immediate effect, i.e., the source shall not start using the protocol suite configuration of the modified contract as soon as it receives the response along with this configuration information from the SCO. This is because the SCO still needs to increase or change the allocation of network resources. The response from the SCO shall include an Activation\_Time output argument that indicates to the source when it can start using the new protocol suite configuration. This results in a delayed effect.

#### **6.3.11.2.11.2 Devices that can modify contracts**

Only the source or the system manager shall have the ability to modify an existing contract of this source.

#### **6.3.11.2.11.3 Contract modification request and response arguments**

If the source decides to modify a contract, it shall send a contract modification request to the SCO. The SCO shall then send a response back to the source informing it that the contract has been modified. The request shall be an Execute.Request to the SCO, and the response shall be an Execute.Response message. The input and output arguments are provided in Table 27. The SCO may also communicate with relevant devices to allocate or de-allocate the necessary network resources.

If the system manager decides to modify a contract, it shall send a contract modification command to the DMO of the source by using the Modify\_Contract method. The DMO shall then send a response back with the status. The SCO may also communicate with relevant devices to allocate or de-allocate the necessary network resources.

The DMO method to modify an existing contract is described in Table 34.

**Table 34 – DMO method to modify contract**

<b>Standard object type name: Device management object (DMO)</b>		
<b>Standard object type identifier: 127</b>		
<b>Defining organization: ISA</b>		
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>
Modify_Contract	2	Method to modify an existing contract in the Contracts_Table attribute in Table 10. This method uses the Write_Row method template defined in Table 519 with the following arguments: Attribute_ID: 26 (Contracts_Table) Index_1: 1 (Contract_ID)

#### **6.3.11.2.11.4 Contract renewal**

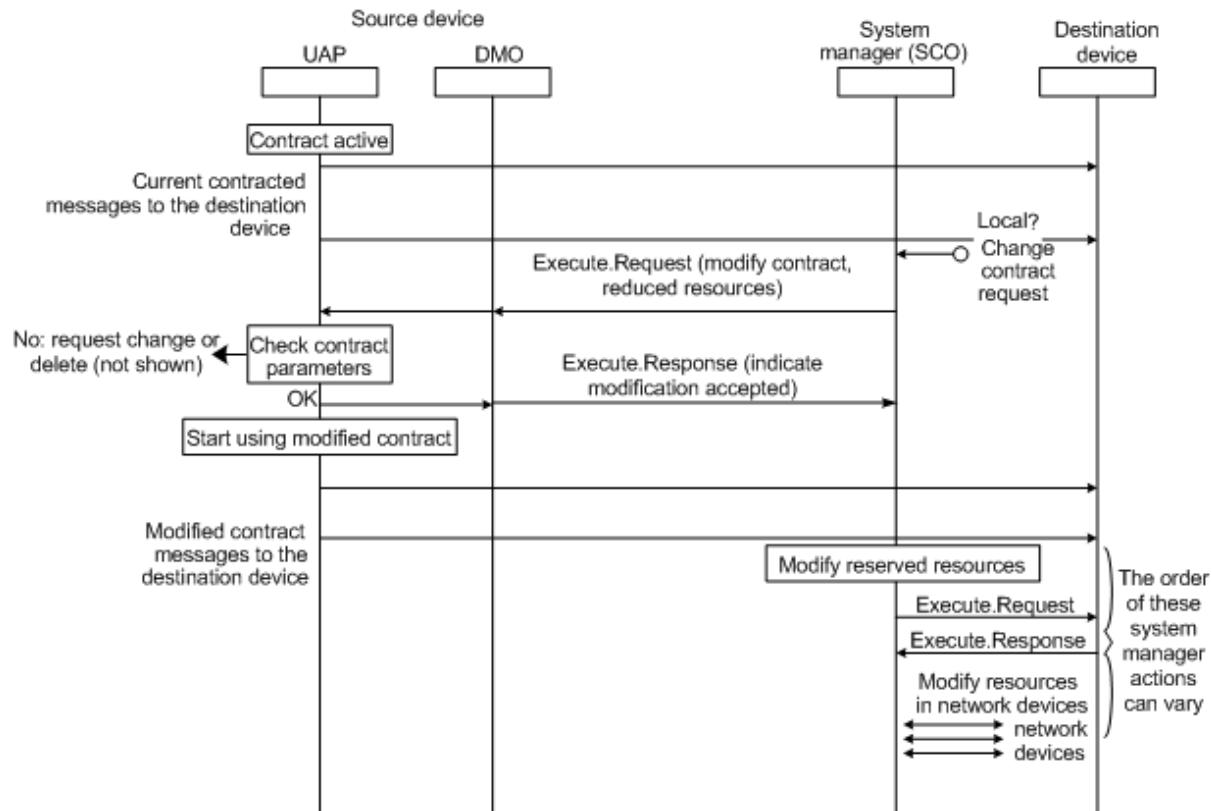
Contract renewal is equivalent to a simple contract modification, with only the Contract\_Expiration\_Time input argument being updated and all other input arguments being the same as those in the original contract request.

#### **6.3.11.2.11.5 Protocol suite configuration**

As part of contract modification, the SCO shall configure / re-configure the necessary devices in the network by providing necessary protocol suite configurations to each of them. This shall include the re-configuration of the destination and the source. Procedures similar to the ones used for protocol suite configuration during contract establishment (see 6.3.11.2.6) may be used by the SCO for this purpose.

#### **6.3.11.2.11.6 Contract modification message sequence diagram**

Figure 33 shows the message sequence chart for modifying a contract with immediate effect.



**Figure 33 – Contract modification with immediate effect**

#### 6.3.11.2.11.7 Contract modification and session key updates

Session key updates are not treated as contract modifications. Such key updates shall be sent from the security manager, through the proxy security management object (PSMO) in the system manager, to the relevant device pair part of the corresponding session.

#### 6.3.11.2.12 Contract failure scenarios

Table 27 contains output arguments for failure scenarios in which the system manager is not able to support the contract request. Such failures may occur if the requested communication service cannot be supported at all, if it cannot be supported due to a temporary condition, or if it cannot be supported unless the request is resent by the source with arguments negotiated down. In such cases, the system manager may choose to include output arguments in the response that provide some guidance to the source. These include `Retry_Backoff_Time` and `Negotiation_Guidance`.

#### 6.3.12 Redundancy management

Although this standard incorporates features for simplex and optional full redundancy from field devices to plant network, redundancy management is not specified in this standard.

The system manager is expected to be capable of configuring path redundancy in the DL subnet through the field routers. Field devices including field routers can be configured to communicate with redundant backbone routers.

Device-level redundancy that requires synchronization between the redundant devices to maintain state information is allowed, but is not specified in this standard.

#### 6.3.13 System management protocols

Management-related communication between devices compliant with this standard and the system manager shall be accomplished via standard application sub-layer messaging, as described in 12.12.

**6.3.14 Management policy administration**

Management policies and policy administration is not specified by this standard. A default policy may be established to make all device information available to the system manager (with appropriate security). Overview information may be made available outside the network (e.g., the network is operating within nominal limits).

**6.3.15 Operational interaction with plant operations or maintenance personnel**

While the device implementing the system manager may have an interface that allows plant operations and maintenance personnel to observe and control the performance of the network and devices, this interface is neither mandatory nor is it specified by this standard.

## 7 Security

### 7.1 General

This clause describes the security sub-layer functionality, its interface with the data link layer (DL) and the transport layer (TL), and the protection of data in transit. The primary focus of this clause is to provide transmission security and related security aspects including the join process, session establishment, key updates, and associated policies. This standard does not address other types of security, such as security of data-at-rest or physical device security.

The specific messages that are protected are single-hop (hop-by-hop) DPDUs, end-to-end transport TPDUs, and security management data structures. A steady-state data flow using DPDUs and TPDUs that may be protected is outlined in Figure 34. The endpoints in a TL security association are defined by the end device as well as the end application.

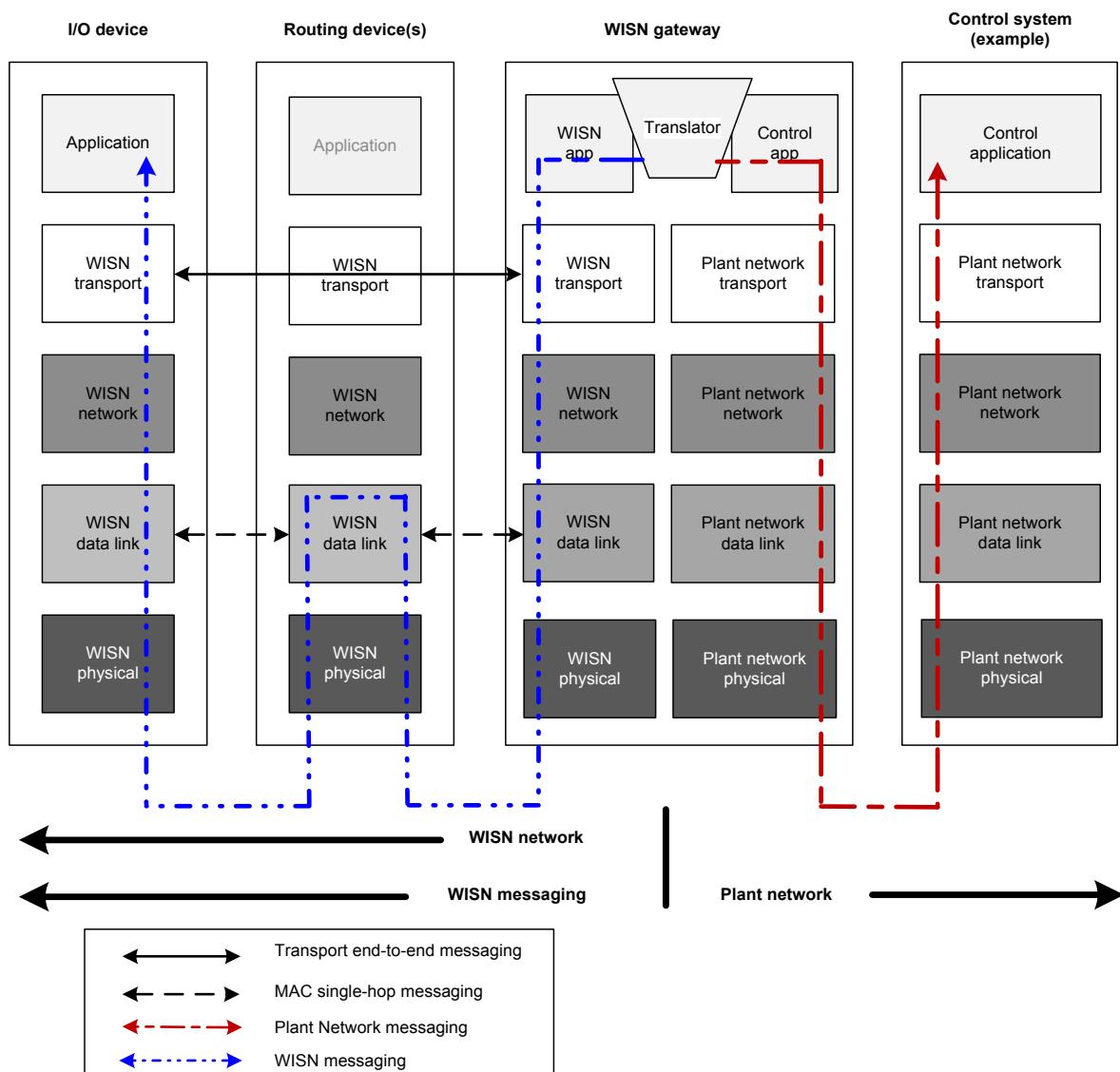


Figure 34 – Examples of DPDU and TPDU scope

### 7.2 Security services

#### 7.2.1 Overview

The security services in this standard are selected by policy. The policy is distributed with each cryptographic material, permitting focused policy application. Since a single key is used at a time at the DL, except for a brief period of the key handover, the entire sub-network is

subject to the same policies at the DL. The security manager controls the policies for all the cryptographic materials it generates.

Devices with appropriate credentials participate in secured communications with other such devices through the use of a shared-secret symmetric key used to authenticate and optionally encrypt their messages to each other.

The security services are applied at the bottom of the communication protocol stack, hop-by-hop at the DL, and at the top of the communication protocol stack, end-to-end at the TL. Security management services are also used by the application layer for the join process, key distribution and session management. When secret keys are used, DL security defends against attackers who are outside the system and do not share system secrets, while TL security defends against attackers who may be on the network path between the source and the destination.

In both cases, a symmetric data key shared among intended communicants is used to add a keyed integrity check to the message and to provide optional encryption of the payload carried by the message. Attackers that do not share the key cannot modify the message without being detected and cannot decrypt the encrypted payload information.

The security operation is based on a shared sense of time that usually is aligned with the TAI time (see 5.6). The DL and TL security services authenticate to the receiver using the TAI time of DPDU transmission and the approximate time of TPDU creation.

When three or more devices share a common secret key, source authentication is no longer guaranteed within that group because of the shared symmetric key. In this case, intra-group source authentication requires complex mechanisms; thus, authentication of the specific sending node (within the multicast group) is not addressed.

The primary security components of the provided services include:

- Authorization of secure communications relationships between entities;
- Message authenticity, ensuring that messages originate from an authorized member of a communications relationship and that they have not been modified by an entity outside of the relationship between the sender and the receiver;
- Assurance that delivery timing and order does not exceed anticipated bounds;
- Data confidentiality that conceals the contents (other than size) among message payloads; and
- Protection against malicious replay attack.

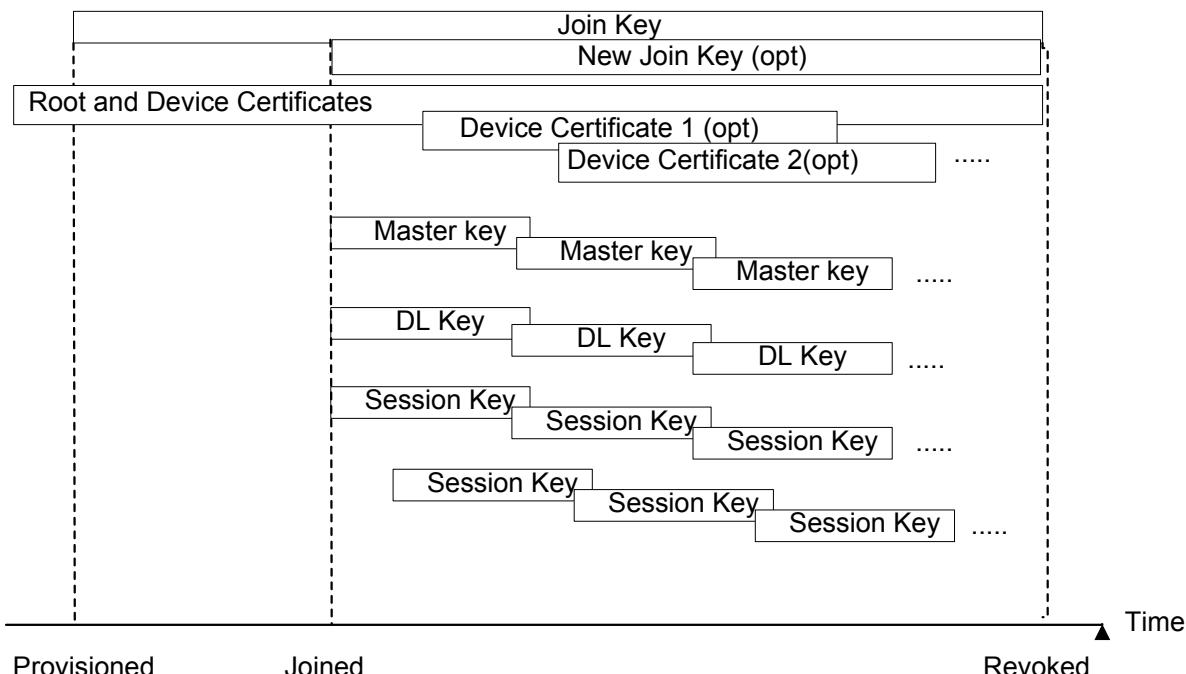
Various combinations of these services are provided at both the DL and the TL. Additionally, various cryptographic services are available for use by the DSMO for the join process, session establishment and key update.

NOTE Protection against compromise of the cryptographic boundaries inside the hardware of devices compliant with this standard is outside of the scope of this standard. Other publications, including ISO/IEC 15408 and the U.S. NIST FIPS 140 series, address those issues. Compliance decisions are left to those who evaluate devices.

## 7.2.2 Keys

### 7.2.2.1 General

Symmetric keys are used for data encryption and authentication; see 7.3.2.5, 7.3.2.6, 7.3.3.8, and 7.3.3.9. Asymmetric keys can be used for the join process, see 7.4. Each key is limited in time and can be updated. Figure 35 shows the types of keys specified by this standard and their associated lifetimes. It also shows an optional asymmetric-key security certificate.



**Figure 35 – Keys and associated lifetimes**

#### 7.2.2.2 Symmetric keys

All WISN symmetric keys shall be of 128 bits. The symmetric keys used shall include:

- Global key: a well-known key that shall not be used to guarantee any security properties.
- K\_open: This key is used as the join key in the provisioning step described in 14.3. The actual value for this key is 0x004F00500045004E0000000000000000, which is the representation of the null terminated Unicode string “OPEN” followed by trailing 0x00 octets. The crypto key identifier for this key is 1.
- K\_global: This key is used as the join key in the provisioning phase, and as the DL key in the joining phase. Use of this key in the provisioning phase is described in 14.3. The actual value for this key is 0x00490053004100200031003000300000, which is the representation of the null terminated Unicode string “ISA(space)100(null)”. The crypto key identifier for this key is 0.
- Join key (K\_join): A key received at the conclusion of the provisioning step. It is used to join a network. See 3.3.1.5. The K\_global is used as default value of the K\_join.
- Master key: A key first derived at the conclusion of the key agreement scheme. That key is used for communication between the security manager and the device. It expires and needs to be periodically updated.
- DL key: A key used to encrypt/decrypt and/or authenticate PDUs at the DL. That key expires and needs to be periodically updated.
- Session key: A key used to encrypt/decrypt and/or authenticate PDUs at the transport layer. That key expires and needs to be periodically updated.

#### 7.2.2.3 Asymmetric keys and certificates

All WISN asymmetric keys shall have cryptographic bit strength of 128 bits. The asymmetric keys used shall include:

- CA\_root: The public key of a certificate authority that signed a device’s asymmetric-key certificate. This key is commonly referred to as a root key and is used to assist in verifying the true identity of the device communicating the certificate, as well as some related keying information.

- Cert-A: The asymmetric-key certificate of device A, used to evidence the true identity of the device, as well as related keying information, during execution of an authenticated asymmetric-key key establishment protocol.

The description of the asymmetric key cryptographic material is provided in H.3.

#### 7.2.2.4 Key lifetime

##### 7.2.2.4.1 General

Symmetric keys are limited by a lifetime and they should be invalidated after the lifetime expiration. To keep a secure ongoing communication, the current keys are updated. In this specification, the key lifetimes (and related information) are defined as follows:

- **ValidNotAfter**  
Absolute TAI time after which a key becomes invalid;
- **SoftExpirationTime**  
Absolute TAI time that a device starts to prepare for updating a key;
- **HardLifeSpan**  
Relative duration from ValidNotBefore to ValidNotAfter;
- **SoftLifeSpan**  
Relative duration from ValidNotBefore to SoftExpirationTime;
- **ValidNotBefore**  
Absolute TAI time at which a key will be enabled; and
- **KeyExchangeMargin**  
Minimum time required to complete a key update cycle.

The relationship of above lifetime definitions is illustrated in Figure 36. The key update mechanism using those time definitions is described in 7.6.

The special value 0 is used for infinite lifetime. The infinite lifetime is used for Global keys specified in 7.2.2.2.

NOTE 1 It is not recommended to have an infinite lifetime for DL, TL, and Master key. An infinite key without periodic key updating makes a system vulnerable.

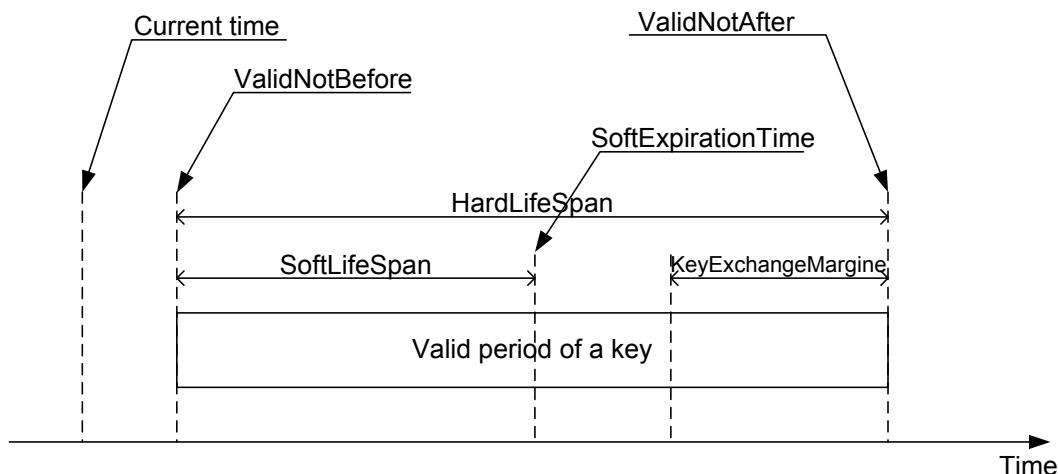


Figure 36 – Key lifetimes

NOTE 2 If a key is used after its hard lifetime, the communication becomes vulnerable to replay attacks and is no longer secure.

NOTE 3 Asymmetric key certificates should have a lifetime (ValidNotBefore and ValidNotAfter), as defined in 7.4.6.2.1.1.

NOTE 4 KeyExchangeMargin can be used as a trigger for invoking the PSMO.Key\_Update\_Request() method to keep the continuous secure session. It is recommended that KeyExchangeMargin is set to "5 times DSMO.pduMaxAge" seconds. It consists of:

- 2 \* DSMO.pduMaxAge seconds for a Security\_New\_Session() method round-trip communication;
- 2 \* DSMO.pduMaxAge seconds for a New\_Key() method round-trip communication and;
- Another DSMO.pduMaxAge seconds for the processing time.

#### **7.2.2.4.2 Key lifetime expiration**

##### **7.2.2.4.2.1 SoftExpirationTime**

When the SoftLifetime expires, the device owning the key prepares to get a new key from the security manager. The device may optionally call the PSMO.Security\_New\_Session() method on the system manager to explicitly request a key, or it can wait to have its DSMO.New\_Key() method called by the security manager. If the device wants to be certain about updating a key, it should call the PSMO.Security\_New\_Key() on the system manager method explicitly.

It is not necessary for the device to start the key update process immediately after the SoftLifetime expires. The key update can be accomplished at any time up to the ValidNotAfter time. To keep the current secure session with a peer, a request for a new key can be issued half way between SoftExpirationTime and ValidNotAfter. The device should call the PSMO.Security\_New\_Key() method before the time (HardExpirationTime – KeyExchangeMargin).

##### **7.2.2.4.2.2 ValidNotAfter**

If the ValidNotAfter time expires, the key shall not be used in active communication subsequently and should be removed by device owing the key, but the key can be saved by the security manager depending on the key archiving policies.

### **7.3 Frame security**

#### **7.3.1 General**

##### **7.3.1.1 Security level**

The security level specifies the method to be applied to certain PDUs. The security level consists of a combination of the MIC length (0bit, 32bit, 64bit and 128bit) and the encryption Boolean (encrypted or not encrypted). Table 35 shows security levels used in this specification.

**Table 35 – Security levels**

<b>Security level identifier</b>	<b>Security control field <math>b_2b_1b_0</math></b>	<b>Security attributes</b>
0x00	000	None
0x01	001	MIC-32
0x02	010	MIC-64
0x03	011	MIC-128
0x04	100	ENC
0x05	101	ENC-MIC-32
0x06	110	ENC-MIC-64
0x07	111	ENC-MIC-128

##### **7.3.1.2 Security control field**

The security control field is part of each DL and TL security header. It specifies the presence of the key identifier and security level to be applied to the PDU. The security control field octets shall be populated as specified in IEEE Std 802.15.4-2006, 7.6.2.2.

Table 36 shows the structure of the security control field.

**Table 36 – Structure of the security control field**

Octet	Bits							
	7	6	5	4	3	2	1	0
1	Reserved			Crypto Key Identifier Mode			Security level	

The Crypto Key Identifier Mode field represents the length of the following Crypto Key Identifier field immediately after the Security\_Control field. If the Key Identifier Mode is set to 0 ( $b_4b_3$ , '00'), the following Crypto Key Identifier field should be elided.

The security level field shall consist of 3 bits and shall be as defined in IEEE Std 802.15.4-2006, Table 95, and outlined in Table 35 within this standard for reference. The security levels of 0x04, corresponding to encryption only, shall never be used for a data frame in the data link or transport layer in this standard. The security level of 0x00, corresponding to no protection shall never be used for data frame in the data link layer in this standard.

NOTE Encryption only does not provide any secure protection against an active attacker. Such an attacker can arbitrarily modify a packet in transit. Since CTR (counter) mode is used for encryption in CCM\*, the attacker can target specific bits in an encrypted payload wherein bits are flipped in the ciphertext at desired portions and the corresponding bits will be flipped in the decrypted plaintext. Without an integrity field, there is no secure method for the recipient to detect such a change, and thus the recipient will be forced to process a malicious frame.

### 7.3.2 Data link layer frame security

#### 7.3.2.1 General

The degree to which a device is permitted to participate in a DL network or subnet shall be determined by system policy applied to credentials supplied by the device. Devices without credentials shall be permitted full, limited, or no participation beyond join attempts, as determined by system policy for such devices.

The AES security processing engine is always on at the DL. The details of the cryptographic building blocks are in Annex H. In non-secure mode, the key distributed might have traveled over an insecure channel. When a properly secured secret DL key is used, the following security service shall always be provided:

- DPDU authentication;
- DPDU integrity; and
- Proof that the DPDU was received at the intended time, providing rejection of DPDUs:
- That were not sourced by a device within the network that shares an appropriate data key;
- That were not received at a time for which their reception was intended; or
- That were recorded and maliciously replayed.

NOTE 1 Authorization is implied by the fact that the sending device has knowledge of a shared symmetric data key. When the key is not a shared secret, the authorization extends to all possible devices through the use of the global key. When the key is a shared secret, an inference may be made that the sending device obtained the shared-secret key from a security manager, and that it would have obtained the key only if the security manager's authorization database permitted the resulting protected communications relationship. Such permission usually is based on the device's role. See Device\_Role\_Capability (standard object type identifier 127, attribute identifier 4, in Table 10) for a definition of the roles and their respective bitmap.

NOTE 2 The detection of reception at an inappropriate time renders ineffective MAC message stream attacks based on DPDU delay, reordering, or replay. Reordering can happen if it occurs within a 1 ms window.

NOTE 3 This service uses the sender's time of transmission, the receiver's time of reception, and the fact that MAC transmission and reception are concurrent to ensure that any DPDU received at an unintended time, including DPDU replay or DPDU stream reordering, will be detected and the anachronistic DPDU(s) rejected.

The amount of redundancy used to provide DPDU integrity is selected by policy associated with the relevant data key.

The following DL security service shall be selectable by policy associated with the relevant DL key:

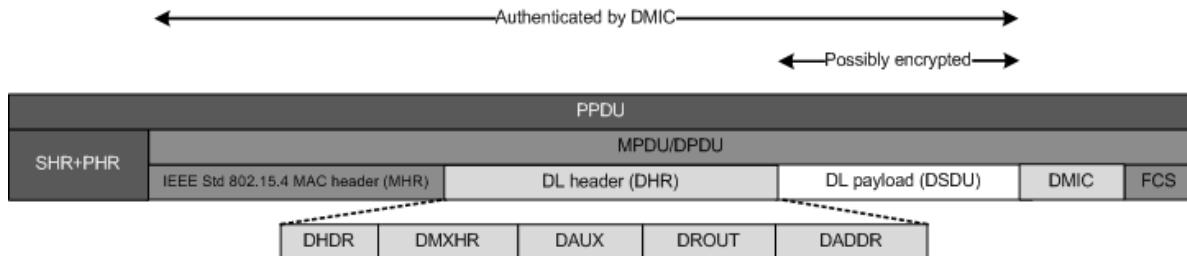
- Encryption (confidentiality) of the DSDU.

This confidentiality service shall not be used with keys that use the values OPEN(null)(null)(null)(null) and ISA(space)100(null), because this would render confidentiality impossible, and because this aspect of the policy associated with such keys is constant.

NOTE 4 The character “(null)” is 0x0000 and the character “(space)” is 0x0020 in Unicode.

### 7.3.2.2 Data link layer protocol data unit structure

The structure of a DPDU is described in 9.3.1, outlined in Figure 88 and Figure 37 in this standard, with the DSDU possibly encrypted and the MHR, DHR and DSDU protected by the DMIC.



**Figure 37 – DPDU structure**

The complete DPDU from the start of the MHR to the end of the DSDU shall be protected by the DMIC. Information relevant to the security sub-layer shall be in the DL MAC extension header (DMXHR) as outlined in Table 112, the 8-bit sequence number as outlined in Table 110, and the channel number.

### 7.3.2.3 Data frame headers

#### 7.3.2.3.1 IEEE Std 802.15.4 media access control header

The security of the DL data frame shall be handled in the protocol stack defined by this standard, and is therefore above the IEEE Std 802.15.4 MAC sublayer. The MAC header shall be as defined in 9.3.3.2.

#### 7.3.2.3.2 Data link layer media access control extension header

The DMXHR outlined in Table 112 shall contain 2 fields used by the security layer. The first field shall contain the security control field as outlined in 7.3.1.2. The second field shall contain the Crypto Key Identifier as specified in IEEE Std 802.15.4-2006, 7.6.2.4.2. In the DMXHR, the Crypto Key Identifier shall never be elided with the Crypto Key Identifier Mode = 0.

The default value of the security level for the DL shall be set to b<sub>2</sub>b<sub>1</sub>b<sub>0</sub>,001, corresponding to authentication only with a DMIC size of 32 bits.

For the DPDU processing steps, the following constraints shall be observed:

- The DMIC size shall be at least 32 bits (0x00, ‘none’ and 0x04, ‘ENC’ shall not be used as security level of DPDU);
- 128 bit MICs shall not be used in any DL layer security; and
- Only 32 bit MICs shall be used for the DL ACK/NACK PDUs in any security level.

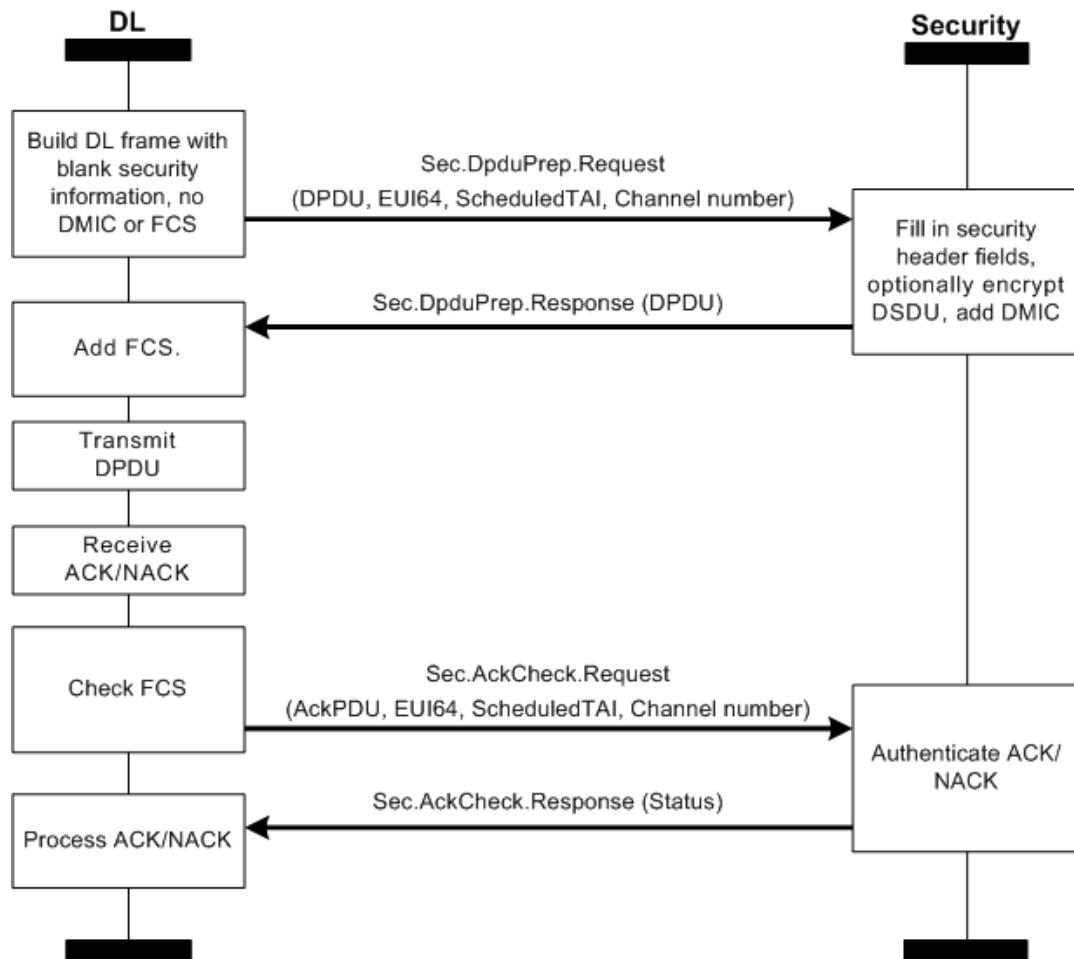
### 7.3.2.4 Interface between the data link layer and security sub-layer

#### 7.3.2.4.1 General

This subclause describes the relationship between the DL and the security sub-layer.

Figure 38 summarizes the relationship between the DL and the security sub-layers for outgoing DPDU. This flow covers the normal case where a DPDU is transmitted and acknowledged, and no errors occur. For more detail, see the documentation for the

corresponding SAPs in 7.3.2.4.2, 7.3.2.4.3, 7.3.2.4.4, 7.3.2.4.5, 7.3.2.4.6, 7.3.2.4.7, 7.3.2.4.8, and 7.3.2.4.9.



**Figure 38 – Outgoing messages – DL and security**

The DL assembles the DPDUs to be protected. By documentation convention, security fields in the DL header are populated by the security sub-layer.

Certain DPDUs security information is provided by the DL to the security sub-layer:

- The scheduled TAI time of the timeslot is provided by the DL to the security sub-layer, and is used in building the nonce. The scheduled TAI time passed to the security sub-layer is the exact start time of the timeslot in which the DPDUs were intended to be transmitted. The security sub-layer truncates this time to  $2^{-10}$  s (approximately 1 ms).
- The channel number of each DPDUs is provided by the DL to the security sub-layer, and is used in building the nonce. The channel number is following the convention in this standard where the numbers 0-15 correspond to IEEE Std 802.15.4 channels 11-26 respectively.
- The one-octet sequence number in the MAC header is used in the nonce to discriminate among multiple DPDUs and ACK/NACKs that a device generates with the same scheduled start time. (See Note 2 below.) . The MHR sequence number octet is unique and ACKs that a given device transmits corresponding to any scheduled timeslot time truncated to  $2^{-10}$  s.
- The security sub-layer needs the EUI-64 of the destination device in order to process its acknowledgement. This is provided by the system manager through the dlmo.Neighbor table. If a unicast destination's EUI-64 is not known to the DL, the request EUI-64 indicator in the DHDR frame control octet (Table 111) shall be set to 1, which triggers the destination to return its EUI-64 in the acknowledgement.

NOTE 1 The DSDU length is needed by the security sub-layer to support encrypting the DSDU without encrypting the DPDU header. (The DMIC covers the entire DPDU.) This detail is not shown in Figure 38 or Figure 39.

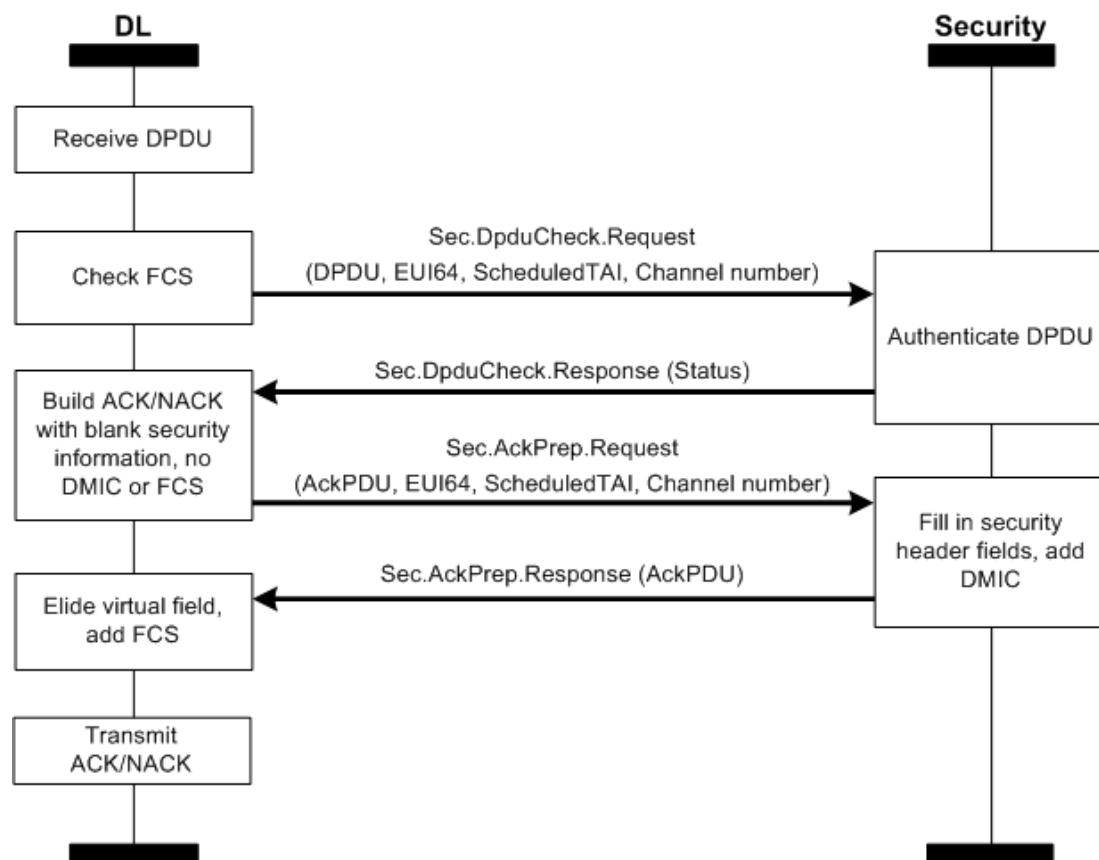
NOTE 2 There are two cases in the standard where a single device might initiate more than one transmission corresponding to a single scheduled timeslot start time. In a burst advertisement, multiple DPDUs are transmitted on one channel in succession, and this case is covered by the sequence number that is incremented with each use as specified by IEEE. The other case is a device operating on multiple channels simultaneously, which is not specified by the standard but is also not prohibited.

The DL retains a copy of the outgoing DMIC, to be used subsequently to unambiguously connect the ACK/NACK to the DPDU. The DL then appends an IEEE Std 802.15.4 FCS and transmits the DPDU.

When the DL receives an ACK/NACK, it requests the security sub-layer to authenticate the DPDU. Certain DPDU security information is provided by the DL to the security sub-layer:

- The ACK/NACK shall echo the DPDU's DMIC as a virtual field (see Table 117). The full ACK/NACK, including this virtual field, is reconstructed by the DL before it is checked by the security sub-layer.
- The EUI-64 of the ACK/NACK originator is either looked up or provided within the ACK/NACK itself.
- Scheduled timeslot time for the ACK/NACK, which is usually the same as the timeslot for the DPDU. However, when slow hopping is used, it is possible that the acknowledgement will include a timeslot offset (see 9.3.4). In this case, the ACK's nonce shall use the scheduled start time of the timeslot referenced by the timeslot offset; that is, the scheduled timeslot of the acknowledging device.
- The channel number for sending the ACK/NACK is provided to the security sub-layer.
- MHR sequence number is provided to the security sub-layer in the same manner that it is provided for the DPDU.

Figure 39 illustrates the relationship between the DL and the security layer for incoming DPDUs.



**Figure 39 – Incoming messages- DL and security**

When receiving a DPDU, the DL sends a request to the security sub-layer to authenticate the DPDU and, if necessary, decrypt the DPDU payload. The scheduled TAI time and the channel number are included with this request. The EUI-64 of the DPDU's source needs to be known by the DL a priori, except in the case of a join request where it is carried in the DPDU header as a source address. The security sub-layer normally responds with a positive authentication.

The DL then builds the ACK/NACK. The ACK/NACK shall use the same scheduled TAI time as the received DPDU, except when a slow hopping offset correction is provided in the ACK/NACK as discussed above. The ACK/NACK shall also echo the received DPDU's DMIC as a virtual field, as shown in Table 117. The security sub-layer then secures the ACK/NACK, including the DPDU's DMIC as a virtual field. The DL elides the virtual field, appends an IEEE Std 802.15.4 FCS, and transmits the ACK/NACK.

When a DL's time is corrected by an acknowledgement, such that time is reset to an earlier timeslot, there shall be an interruption in service, equal to the magnitude of the timeslot correction plus at least one timeslot.

#### **7.3.2.4.2 Sec.DpduPrep.Request**

##### **7.3.2.4.2.1 General**

Sec.DpduPrep.Request instructs the security sub-layer to protect a DL protocol data unit as appropriate.

##### **7.3.2.4.2.2 Semantics of the service primitive**

The semantics of Sec.DpduPrep.Request are as follows:

```
Sec.DpduPrep.Request (DPDU,
                      EUI64,
                      ScheduledTAI,
                      ChannelNumber,
                      AckHandle)
```

Table 37 specifies the elements for the Sec.DpduPrep.Request.

**Table 37 – Sec.DpduPrep.Request elements**

<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
DPDU (The DPDU to be transmitted)	1	Type: Data
EUI64 (The EUI-64 of the sending device)	2	Type: EUI64 Valid value set: N/A
ScheduledTAI (32-bits of the start time of the slot truncated to a $2^{-10}$ s resolution)	3	Type: UInt32 Valid value set: 0- $2^{32}$
ChannelNumber (The channel number used in the transmitted DPDU)	4	Type: UInt8 Valid value set: 0-15
AckHandle (Abstraction that connects each invocation of DIAckCheck.Request with the subsequent callback by DIAckCheck.Response)	5	Type: Abstract

The security sub layer shall populate the DMXHR (the format of which is described in 9.3.3.4) of the DPDU with the appropriate security control (octet 1) and the Crypto Key Identifier (octet 2) obtained from the KeyDescriptor for the current DL key. See 7.3.2.5 on selecting the proper DL key.

The security sub-layer shall populate the DMIC field as dictated by the current policies.

##### **7.3.2.4.2.3 Appropriate usage**

The DL invokes the Sec.DpduPrep.Request primitive to protect a DPDU before it is transmitted.

##### **7.3.2.4.2.4 Effect on receipt**

On receipt of the Sec.DpduPrep.Request primitive, the security sub-layer starts the appropriate PDU processing steps to protect the DPDU as dictated by policy.

### 7.3.2.4.3 Sec.D pduPrep.Response

#### 7.3.2.4.3.1 General

Sec.D pduPrep.Response reports the result of a Sec.D pduPrep.Request.

#### 7.3.2.4.3.2 Semantics

The semantics of Sec.D pduPrep.Response are as follows:

Sec.D pduPrep.Response	(
	DPDU,
	Status,
	AckHandle)

Table 38 specifies the elements for Sec.D pduPrep.Response.

**Table 38 – Sec.D pduPrep.Response elements**

Element name	Element identifier	Element scalar type
DPDU	1	Type: Data Valid value set: N/A
Status (The result of a Sec.D pduPrep.Request primitive)	2	Type: Enumeration Valid value set: SUCCESS, FAILURE
AckHandle (Abstraction that connects each invocation of DIAckCheck.Request with the subsequent callback by DIAckCheck.Response)	3	Type: Abstract

#### 7.3.2.4.3.3 When generated

The security sub-layer generates Sec.D pduPrep.Response in response to a Sec.D pduPrep.Request. The Sec.D pduPrep.Response returns a status value that indicates either SUCCESS and the unsecured DPDU or the appropriate error code.

#### 7.3.2.4.3.4 Appropriate usage

On receipt of Sec.D pduPrep.Response, the data link layer is notified of the result of request to protect an outgoing DPDU.

### 7.3.2.4.4 Sec.DL AckCheck.Request

#### 7.3.2.4.4.1 General

Sec.DL AckCheck.Request instructs the security sub-layer to verify an incoming DL ACK protocol data unit.

#### 7.3.2.4.4.2 Semantics of the service primitive

The semantics of Sec.DL AckCheck.Request are as follows:

Sec.DL AckCheck.Request	(
	AckPDU,
	EUI64,
	ScheduledTAI,
	ChannelNumber,
	AckHandle)

Table 39 specifies the elements for the Sec.AckCheck.Request.

**Table 39 – Sec.DLAckCheck.Request elements**

Element name	Element identifier	Element scalar type
AckPDU (The AckPDU to be verified)	1	Type: Data Valid value set: N/A
EUI64 (The EUI-64 of the acknowledging device)	2	Type: EUI64 Valid value set: N/A
ScheduledTAI (32-bits of the start time of the slot truncated to a $2^{-10}$ s resolution)	3	Type: UInt32 Valid value set: 0- $2^{32}$
ChannelNumber (The channel number used to receive the incoming ACK/NACK PDU)	4	Type: UInt8 Valid value set: 0-15
AckHandle (Abstraction that connects each invocation of DIAckCheck.Request with the subsequent callback by DIAckCheck.Response)	5	Type: Abstract

The security sub-layer shall verify that the DHR of the ACK/NACK PDU has the appropriate DMIC option (see Table 118) by comparing it to the current policy. The key used to authenticate the ACK/NACK shall be the same as used by the outgoing DPDU.

The security sub-layer shall verify the DMIC field as dictated by the PDU processing steps and current policies.

#### 7.3.2.4.4.3 Appropriate usage

The DL invokes the Sec.DLAckCheck.Request primitive to verify a DL ACK/NACK PDU after reception.

#### 7.3.2.4.4.4 Effect on receipt

On receipt of the Sec.DLAckCheck.Request primitive, the security sub-layer starts the appropriate PDU processing steps to verify the incoming DL ACK/NACK PDU as dictated by the incoming PDU processing steps in 7.3.2.6.

#### 7.3.2.4.5 Sec.DLAckCheck.Response

##### 7.3.2.4.5.1 General

Sec.DLAckCheck.Response reports the result of a Sec.DLAckCheck.Request.

##### 7.3.2.4.5.2 Semantics

The semantics of Sec.DLAckCheck.Response are as follows:

Sec.DpduCheck.Response (AckPDU,  
Status,  
AckHandle)

Table 40 specifies the elements for Sec.DLAckCheck.Response.

**Table 40 – Sec. DLAckCheck.Response elements**

Element name	Element identifier	Element scalar type
AckPDU	1	Type: Data Valid value set: N/A
Status (The result of a Sec.DLAckPrep.Request primitive)	2	Type: Enumeration Valid value set: SUCCESS, FAILURE
AckHandle (Abstraction that connects each invocation of DIAckCheck.Request with the subsequent callback by DIAckCheck.Response)	3	Type: Abstract

##### 7.3.2.4.5.3 When generated

The security sub-layer generates Sec.DLAckCheck.Response in response to a Sec.DLAckCheck.Request. The Sec.DLAckCheck.Response returns a status value that indicates either SUCCESS or the appropriate error code.

#### **7.3.2.4.5.4 Appropriate usage**

On receipt of Sec.DLAckCheck.Response, the data link layer is notified of the result of verifying and possibly decrypting an incoming DPDU.

#### **7.3.2.4.6 Sec.DpduCheck.Request**

##### **7.3.2.4.6.1 General**

Sec.DpduCheck.Request instructs the security sub-layer to verify and possibly decrypt an incoming DL protocol data unit as appropriate.

##### **7.3.2.4.6.2 Semantics of the service primitive**

The semantics of Sec.DpduCheck.Request are as follows:

```
Sec.DpduCheck.Request (DPDU,
EUI64,
ScheduledTAI,
ChannelNumber,
AckHandle)
```

Table 41 specifies the elements for the Sec.DpduCheck.Request.

**Table 41 – Sec.DpduCheck.Request elements**

Element name	Element identifier	Element scalar type
DPDU (The DPDU to be verified and possibly decrypted)	1	Type: Data
EUI64 (The EUI-64 of the sending device)	2	Type: EUI64 Valid value set: N/A
ScheduledTAI (32-bits of the start time of the slot truncated to a $2^{-10}$ s resolution)	3	Type: UInt32 Valid value set: 0- $2^{32}$
ChannelNumber (The channel number used to receive the incoming DPDU)	4	Type: UInt8 Valid value set: 0-15
AckHandle (Abstraction that connects each invocation of DIAckCheck.Request with the subsequent callback by DIAckCheck.Response)	5	Type: Abstract

The security sub-layer shall verify that the DMXHR of the DPDU has the appropriate security control (octet 1) by comparing it to the current policy. The Crypto Key Identifier (octet 2) shall be used to retrieve the correct key material. See 7.3.2.6

The security sub-layer shall verify the DMIC field as dictated by the current policies.

##### **7.3.2.4.6.3 Appropriate usage**

The DL invokes the Sec.DpduCheck.Request primitive to verify and possibly decrypt a DPDU before it is transmitted.

##### **7.3.2.4.6.4 Effect on receipt**

On receipt of the Sec.DpduCheck.Request primitive, the security sub-layer starts the appropriate PDU processing steps to verify the incoming DPDU as dictated by the incoming PDU processing steps in 7.3.2.6.

#### **7.3.2.4.7 Sec.DpduCheck.Response**

##### **7.3.2.4.7.1 General**

Sec.DpduCheck.Response reports the result of a Sec.DpduCheck.Request.

##### **7.3.2.4.7.2 Semantics**

The semantics of Sec.DpduCheck.Response are as follows:

```
Sec.D pduCheck.Response
  (
    DPDU,
    Status,
    AckHandle)
```

Table 42 specifies the elements for Sec.D pduCheck.Response.

**Table 42 – Sec.D pduCheck.Response elements**

Element name	Element identifier	Element scalar type
DPDU	1	Type: Data Valid value set: N/A
Status (The result of a Sec.D pduPrep.Request primitive)	2	Type: Enumeration Valid value set: SUCCESS, FAILURE
AckHandle (Abstraction that connects each invocation of DIAckCheck.Request with the subsequent callback by DIAckCheck.Response)	3	Type: Abstract

#### 7.3.2.4.7.3 When generated

The security sub-layer generates Sec.D pduCheck.Response in response to a Sec.D pduCheck.Request. The Sec.D pduCheck.Response returns a status value that indicates either SUCCESS or the appropriate error code.

#### 7.3.2.4.7.4 Appropriate usage

On receipt of Sec.D pduCheck.Response, the data link layer is notified of the result of verifying and possibly decrypting an incoming DPDU.

#### 7.3.2.4.8 Sec.DLAckPrep.Request

##### 7.3.2.4.8.1 General

Sec.DLAckPrep.Request instructs the security sub-layer to protect a DL ACK/NACK as appropriate.

##### 7.3.2.4.8.2 Semantics of the service primitive

The semantics of Sec.DLAckPrep.Request are as follows:

```
Sec.DLAckPrep.Request
  (
    AckPDU,
    EUI64,
    ScheduledTAI,
    ChannelNumber,
    AckHandle)
```

Table 43 specifies the elements for the Sec.DLAckPrep.Request.

**Table 43 – Sec.DLAckPrep.Request elements**

Element name	Element identifier	Element scalar type
AckPDU (Includes the virtual header)	1	Type: Data
EUI64 (The EUI-64 of the acknowledging device)	2	Type: EUI64 Valid value set: N/A
ScheduledTAI (32-bits of the start time of the slot truncated to a $2^{-10}$ s resolution)	3	Type: UInt32 Valid value set: 0-2 <sup>32</sup>
ChannelNumber (The channel number used to transmit the ACK/NACK PDU)	4	Type: UInt8 Valid value set: 0-15
AckHandle (Abstraction that connects each invocation of DIAckCheck.Request with the subsequent callback by DIAckCheck.Response)	5	Type: Abstract

The security sub-layer shall populate the DL ACK/NACK of the DPDU with the appropriate security control (octet 1) as described in Table 118. In the case where multiple DL keys are currently valid, the key used to authenticate the ACK/NACK PDU shall be the same one used as the corresponding DPDU for this ACK/NACK PDU.

The security sub-layer shall populate the DMIC field as dictated by the current policies. Note that only a DMIC length of 32 bits is allowed on the ACK/NACK frame.

#### 7.3.2.4.8.3 Appropriate usage

The DL invokes the Sec.DLAckPrep.Request primitive to protect a DL ACK/NACK before it is transmitted.

#### 7.3.2.4.8.4 Effect on receipt

On receipt of the Sec.DLAckPrep.Request primitive, the security sub-layer starts the appropriate PDU processing steps to protect the ACK/NACK frame as dictated by policy. Note that the ACK/NACK frame is only authenticated and never encrypted.

#### 7.3.2.4.9 Sec.DLAckPrep.Response

##### 7.3.2.4.9.1 General

Sec.DLAckPrep.Response reports the result of a Sec.DLAckPrep.Request.

##### 7.3.2.4.9.2 Semantics

The semantics of Sec.DLAckPrep.Response are as follows:

Sec.AckPrep.Response (AckPDU,  
Status,  
AckHandle)

Table 44 specifies the elements for Sec.DLAckPrep.Response.

**Table 44 – Sec.DLAckPrep.Response elements**

Element name	Element identifier	Element scalar type
AckPDU	1	Type: Data Valid value set: N/A
Status (The result of a Sec.AckPrep.Request primitive)	2	Type: Enumeration Valid value set: SUCCESS, FAILURE
AckHandle (Abstraction that connects each invocation of DIAckCheck.Request with the subsequent callback by DIAckCheck.Response)	3	Type: Abstract

##### 7.3.2.4.9.3 When generated

The security sub-layer generates Sec.AckPrep.Response in response to a Sec.DLAckPrep.Request. Sec.DLAckPrep.Response returns a status value that indicates either SUCCESS or the appropriate error code.

##### 7.3.2.4.9.4 Appropriate usage

On receipt of Sec.DLAckPrep.Response, the data link layer is notified of the result of request to verify an incoming AckPDU.

#### 7.3.2.4.10 Nonce construction for data link layer protocol data units

This standard uses a different DPDU nonce construction from that of the IEEE Std 802.15.4. A 13-octet nonce is required for the CCM\* engine. The nonce shall be constructed as the concatenation from first (leftmost) to last (rightmost) octets of data fields as shown in Table 45, wherein:

- The EUI-64 shall be used as an array of 8 octets (in MSB convention) in the same manner as the source address of the CCM\* nonce in IEEE Std 802.15.4-2006, 7.6.3.2;
- The TAI time shall be least significant 32 bits of the TAI time in units of  $2^{-10}$  s as described in Table 46;
- The last octet shall be constructed as follows:
- Bit 7 shall be zero, thereby reserving the value 0xFF for the transport layer (See Table 55).

- Bits 6-3 (4 bits) shall indicate the radio channel of transmission, in a range of 0 to 15, corresponding to IEEE Std 802.15.4 channel numbers 11 to 26, in the same order.
- Bits 2-0 shall be copied from the corresponding low-order 3 bits of the MHR's sequence number.

**Table 45 – Structure of the WISN DPDU nonce**

octets	bits							
	7	6	5	4	3	2	1	0
1	EUI-64 address of DPDU originator							
...								
8								
9	Least Significant 32 bits of TAI time of nominal slot start (in units of $2^{-10}$ s)							
...								
12								
13	Reserved = 0	Channel number (0 to 15)				Low order 3 bits of MHR sequence number		

The TAI time used shall be a 32-bit truncated representation of the TAI time at a granularity of  $2^{-10}$  seconds. This allows the representation of  $2^{22}$  seconds. With this representation, there will be 48.5 days before the same value of TAI time recurs. Thus, the maximum lifetime of a DL key shall be 48.5 days before a new key needs to be deployed. The source of TAI time for this operation shall be provided by the DL.

NOTE 1 It is important that the value of the 32-bit representation of the TAI time does not recur to avoid a potential nonce collision with resultant keystream reuse.

The least significant 32-bits of TAI time to the  $2^{-10}$  s field of the nonce is described in Table 46.

**Table 46 – Structure of the 32-bit truncated TAI time**

octets	bits							
	7	6	5	4	3	2	1	0
1	Truncated TAI time (bits with weight $2^{21}$ to $2^{14}$ seconds)							
2	Truncated TAI time (bits with weight $2^{13}$ to $2^6$ seconds)							
3	Truncated TAI time (bits with weight $2^5$ to $2^2$ seconds)							
4	Truncated TAI time (bits with weight $2^{-2}$ to $2^{-10}$ seconds)							

The lower order 3 bits of the MHR sequence number ( $b_2b_1b_0$ ) and the channel number are used to construct the last octet of a nonce. The MHR sequence number is unique across all transmissions by the sending device for an identical 32-bit TAI time value and key material (See 9.3.3.2 and 9.3.4). The value of 0xFF shall not be used for the DL nonce. Because this nonce has at most 8 distinct values for a given channel number, no device on a WISN is permitted to transmit more than 8 DPDUs per  $2^{-10}$  second on the same channel. By utilizing the channel number in a nonce, a device operating simultaneously on multiple channels can be accommodated.

NOTE 2 The construction of sequence number is described in 9.3.3.2.

### 7.3.2.5 Outgoing data link layer protocol data unit processing

The inputs to the DPDU security procedure are:

- 1) The DPDU to be secured
- 2) The EUI-64 of the source device
- 3) The nominal TAI time
- 4) The MHR sequence number octet
- 5) The channel number (0-15) to be used for the outgoing DPDU

The outputs from this procedure are

- 1) The status of the procedure and,
- 2) If this status is success, the secured frame.

The outgoing frame security procedure consists of the following steps:

- 1) The procedure shall obtain the KeyDescriptor from Table 94 meeting the following selection criteria:
    - The entries with KeyUsage = '0x00' (DL key). In the initial case, where a joining device does not have any DL KeyDescriptor, the joining device creates a DL KeyDescriptor with K\_global. The KeyDescriptor shall include at least the following parameters;
      - Crypto Key Identifier = 0
      - Security Level = 0x01 (MIC32 only, no Encryption)
      - KeyUsage = 0x00 (Group key for PDU processing)
      - Key lifetime = infinite (0x0000)
    - Of those entries, the entries valid for the current period, satisfying the inequality "ValidNotBefore < current time < ValidNotAfter" shall be selected. If none are available, the procedure shall return with a status of UNAVAILABLE\_KEY.
    - Of those entries, if two or more keys are valid for the current time, and the procedure was called from an AckPrep.Request or an AckCheck.Request, the procedure shall select the key used to authenticate the DPDU associated with this ACK; otherwise:
      - If two or more keys are valid for the current time, the procedure shall select the key with the longest ValidNotAfter value.
      - Of those entries, if two or more keys have the same ValidNotAfter, the procedure shall select the key with the longest ValidNotBefore.
      - Of those entries, if two or more keys have the same SoftExpirationTime, the procedure shall select the key with the highest Crypto Key Identifier.
- 2) The procedure shall retrieve the policy from the selected KeyDescriptor.
- 3) The procedure shall determine whether the DPDU to be secured satisfies the constraint on the maximum length of DPDUs, as follows:
  - The procedure shall set the length M, in octets, of the DMIC authentication field from the security level.
  - The Crypto Key Identifier Mode field in the DMXHR shall be 1. If the DMXHR includes the slow hopping timeslot offset field, the size the length of DMXHR is 3 octets. Otherwise the length of DMXHR is 2 octets.
  - The procedure shall determine the data expansion as DMXHR size+M.
  - The procedure shall check whether the length of the DPDU to be secured, including data expansion, is less than or equal to the maximum DPDU length. If this check fails, the procedure shall return a status of DPDU\_TOO\_LONG.
- 4) The procedure shall set the TAI time value of the nominal time of DPDU slot start, as described in Table 46. If there is a potential for the device to send multiple DPDUs with the same value of the TAI time, then the procedure shall select a value to be conveyed in the DPDU's MHR header that is different from all other such values originated by the device at this particular value of TAI time. A procedure for populating the sequence number is defined in DL section (9.3.3.2).
- 5) The procedure shall insert the DMXHR into the DPDU outlined in Table 112, with fields set as follows:
  - The security level subfield of the security control field shall be set to the security level, b<sub>2</sub>b<sub>1</sub>b<sub>0</sub>, 001 by default.
  - The Crypto Key Identifier Mode subfield of the security control field shall be set to the Crypto Key Identifier Mode parameter, b<sub>1</sub>b<sub>0</sub>,01 by default.

- 6) The procedure shall set the Crypto Key Identifier octet in the DMXHR. See Table 112.
- 7) The procedure shall insert the MHR sequence number in the Data Frame MHR. See Table 110.
- 8) The procedure shall use the EUI-64 of the transmitting device, the 32 least significant bits of TAI time in  $2^{-10}$  seconds, the lower 3bits of the MHR sequence number, and the channel number to build the nonce as outlined in Table 45.
- 9) The procedure shall use the nonce, the key material, the header, the payload and the CCM\* mode of operation as described in IEEE Std 802.15.4-2006, 7.6.3.4, to secure the DPDU:
  - If the SecurityLevel parameter specifies the use of encryption (see IEEE Std 802.15.4-2006, Table 95), the encryption operation shall be applied only to the DPDU's payload field. The corresponding payload field is passed to the CCM\* transformation process described in IEEE Std 802.15.4-2006, 7.6.3.4, as the unsecured payload. The resulting encrypted payload shall be substituted for the original payload.
  - The remaining fields in the DPDU, up to but not including the payload field, shall be passed to the CCM\* transformation process described in IEEE Std 802.15.4-2006, 7.6.3.4, as the non-payload field.
  - The ordering and exact manner of performing the encryption and integrity operations and the placement of the resulting encrypted data or integrity code within the DPDU payload field shall be as defined in IEEE Std 802.15.4-2006, 7.6.3.4.
- 10) The procedure shall return the secured frame and a status of SUCCESS.

### **7.3.2.6 Incoming data link layer protocol data unit processing**

The input to incoming DPDU security procedure is the DPDU to be unsecured and the channel number on which the DPDU was received. The outputs from this procedure are the unsecured DPDU, the security level, the Crypto Key Identifier Mode, the Crypto Key Identifier, and the status of the procedure. All outputs of this procedure are assumed to be invalid unless and until explicitly set in this procedure. It is assumed that the KeyDescriptors with a single, unique device or a number of devices will have been established by the DSMO.

The security procedure on DPDU reception consists of the following steps:

- 1) The procedure shall set the security level and the Crypto Key Identifier Mode to the corresponding subfields of the security control field of the DMXHR of the incoming DPDU, and the Crypto Key Identifier to the corresponding subfields of the Crypto Key Identifier field of the DMXHR of the DPDU to be unsecured.
- 2) The procedure shall obtain the KeyDescriptor from Table 94 meeting the following selection criteria:
  - The entries with KeyUsage = '0x00' (DL key). In the initial case, where a joining device does not have any DL KeyDescriptors, the joining device creates a temporary DL KeyDescriptor with K\_global. The KeyDescriptor shall include at least following parameters;
    - CryptoKeyIdentifier = 0
    - Security Level = 0x01 (MIC32 only, no encryption)
    - KeyUsage = 0x00 (Group key for PDU processing)
    - Key lifetime = infinite (0x0000)
  - NOTE 1 Device may keep or discard the DL KeyDescriptor for K\_global.
  - NOTE 2 The usage of the DL KeyDescriptor for K\_global is described in 9.1.10.
- Of those entries, the entry with the CryptoKeyIdentifier matching the Crypto Key Identifier of the incoming PDU shall be selected
- If that procedure fails, the procedure shall return with a status of UNAVAILABLE\_KEY.
- 3) The procedure shall determine whether the security level of the incoming DPDU conforms to the security level policy by comparing the SecurityLevel of the matching KeyDescriptor

obtained from step 2 above. If there is a mismatch, the procedure shall return with a status of IMPROPER\_SECURITY\_LEVEL.

- 4) If the lifetime in the KeyDescriptor is finite (> 0x0000), the procedure shall verify that the 8-bit MHR sequence number has not been received previously for the same value of the source EUI-64, the 32-bit TAI time, and the same key. If this check fails, the procedure shall return with a status of DUPLICATE\_DPDU.
- 5) The procedure shall then use the EUI-64 of the sender, the scheduled TAI time, and low order 3 bits of the MHR sequence number, and the channel number to generate the nonce as outlined in Table 45. Additionally, the procedure shall verify that the 8-bit MHR sequence number is not 0xff. If the 8-bit MHR sequence number is 0xff, the procedure shall return with a status of INVALID\_SEQUENCE\_NUMBER.
- 6) The procedure shall use the nonce, the crypto key from the KeyDescriptor obtained in step 2, the actual headers (the non-payload fields), the payload and the MIC of the incoming DPDUs and the CCM\* mode of operation as described in operations (see IEEE Std 802.15.4-2006, 7.6.3.5) to authenticate and optionally decrypt the DPDUs:
  - If the security level specifies the use of encryption (see IEEE Std 802.15.4, Table 98), the decryption operation shall be applied only to the actual DPDUs payload field (see IEEE Std 802.15.4-2006, 7.2.2.2.2). The corresponding payload field shall be passed to the CCM\* inverse transformation process described in IEEE Std 802.15.4-2006, 7.6.3.5 as the secure payload.
  - The remaining fields in the DPDUs shall be passed to the CCM\* inverse transformation process described in IEEE Std 802.15.4-2006, 7.6.3.5 as the non-payload fields (see IEEE Std 802.15.4-2006, Table 100).
  - The ordering and exact manner of performing the decryption and integrity checking operations and the placement of the resulting decrypted data within the DPDUs payload field shall be as defined in IEEE Std 802.15.4-2006, 7.6.3.5.
- 7) If the CCM\* inverse transformation process fails, the procedure shall set the unsecured DPDUs to be the DPDUs to be unsecured and return a status of SECURITY\_ERROR.
- 8) If the lifetime in the KeyDescriptor is not infinite, the procedure shall insert the nonce value (includes MHR sequence number, channel number, source EUI-64, and scheduled TAI time) in the NonceCache field of the corresponding KeyDescriptor, to enable replay protection.
- 9) The procedure shall return with the unsecured DPDUs, the security level, the Crypto Key Identifier Mode, the Crypto Key Identifier, and a status of SUCCESS.

### **7.3.2.7 Detection and discard of duplicated or replayed protocol data units**

See Step 4 in 7.3.2.6.

## **7.3.3 Transport layer security functionality**

### **7.3.3.1 General**

The interaction of the security sub-layer and the transport layer is outlined in this subclause. The transport layer processing steps were written to reuse the commonalities between the DL and the TL. However, since the DL and the TL exist at different network abstraction layers with different requirements and assumptions, there are significant differences between the DL and TL processing steps.

Security services at the transport layer are selected by policy associated with the relevant transport data key, obtained as part of a new session request or a key update and based on transport policy maintained by the security manager associated with any or all of:

- The sending device;
- The requesting UAP; and/or
- The transport association, as defined by its endpoints.

The following transport security service shall always be provided with an active key:

- Authorized communication with TPDU authentication, integrity, and conveyance of the nominal time of TPDU creation, providing rejection of outdated TPDUs;
- That were not sourced by a device within the network that shares an appropriate data key, or
- That were severely outdated, i.e., not received at a time within DSMO.pduMaxAge seconds of the TPDU's nominal time of creation.
- Confidentiality of the application-layer payload within the TPDU.

NOTE 1 Sixteen bits of clock are transmitted with each TPDU.

NOTE 2 This service uses the sender's nominal time of transmission as authenticated at the receiver to cause rejection of TPDUs that are delayed excessively and to provide detection of duplicated TPDUs within that time window.

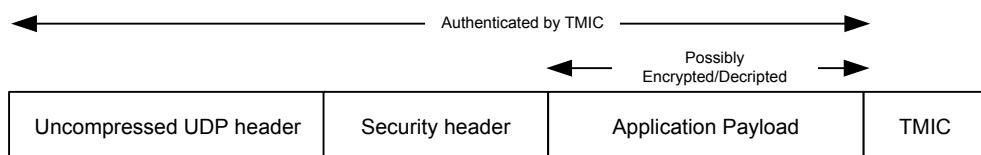
NOTE 3 The confidentiality service is never permitted with keys that are not shared secrets, because this would render true confidentiality impossible, and because this aspect of the policy associated with such keys is constant.

NOTE 4 The MIC should be validated within the DSMO.pduMaxAge period. If the check fails, the MIC validation can be repeated within decrementing a time window to recover the creation time of the PDU.

### 7.3.3.2 Transport layer protocol data unit structure

#### 7.3.3.2.1 General

The structure of a TPDU is described in 11.5, outlined in Figure 109 and in Figure 40 in this standard, with the TSDU possibly encrypted and the contents of the UDP header, security header, and TSDU protected by the TMIC.



**Figure 40 – TPDU structure and protected coverage**

The complete TPDU from the start of the UDP header to the end of the Application Payload shall be protected by the TMIC. Each parameter in the UDP header is protected by using the Transport Security Sub-layer (TSS) pseudo header for the TMIC calculation. The TSS pseudo header is described in 7.3.3.2.2.

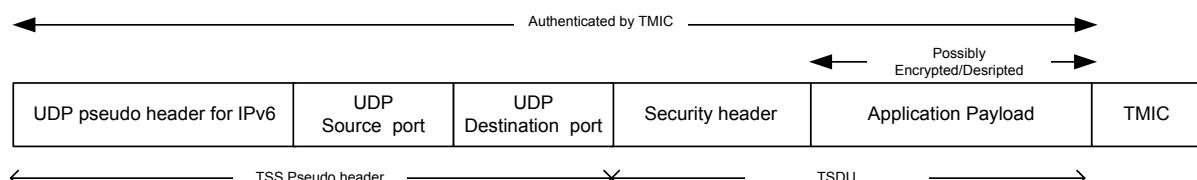
NOTE TSS is described in 11.2.

#### 7.3.3.2.2 TPDU Protection

The TMIC is used to protect the information in the UDP header, the TL security header and the TSDU. Thus, it also protects the Network Layer information including the source and destination IPv6 addresses using the UDP pseudo header for IPv6. The UDP pseudo header for IPv6 is described in 11.4.2. The UDP payload length and the checksum in the UDP header are not used for the TMIC calculation.

NOTE Checksum and UDP payload length do not appear in the pseudo header since the checksum can be elided when the TMIC is present, and the UDP payload length is determined with the NSDU length in the UDP pseudo header for IPv6.

The parameters for the TMIC are shown in Figure 41.



**Figure 41 – TMIC parameters**

The security sub-layer constructs the TMIC parameters as outlined in Figure 41, with the received TPDU, the TAI time at which the TPDU was created the KeyDescriptor and the contract information provided by the Transport layer. The security sub-layer can then use the parameters for the appropriate security operation on the TPDU.

The IPv6 header and the UDP source and destination ports are passed from the transport layer to the security sub-layer. The combination of those parameters is called the TSS pseudo header in this standard. The structure of the TSS pseudo header is shown in Table 47. The appropriate usage is described in 7.3.3.5.4, 7.3.3.5.5, and 7.3.3.5.8.

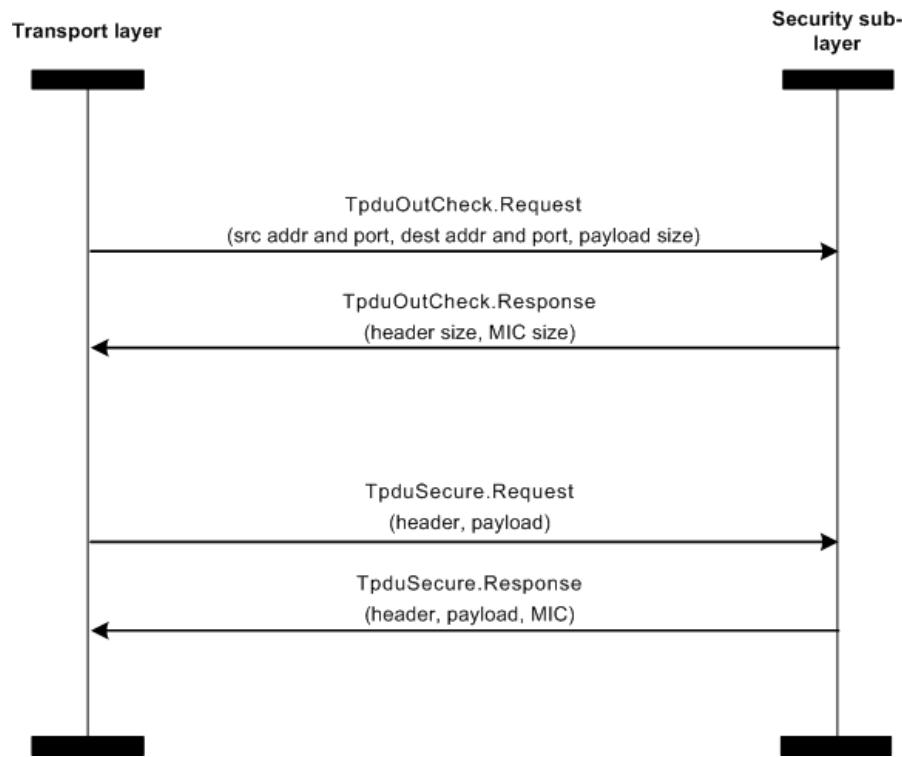
**Table 47 – TSS “pseudo-header” structure**

Element name	Element identifier	Element scalar type
IPv6 Source address	1	Type: Unsigned128 Valid value set: N/A Description: Uncompressed IPv6 address of the TPDU transmitter
IPv6 Destination address	2	Type: Unsigned128 Valid value set: N/A Description: Uncompressed IPv6 address of the TPDU receiver
NSDU length	3	Type: Unsigned16 Valid value set: N/A Description: NDSU length in octet.
Reserved	4	Type: Unsigned8 Valid value set: 0 Description: Reserved field. Currently filled with 0.
Next header	5	Type: Unsigned8 Valid value set: 17 (UDP) Description: Next header value in the IPv6 header. This value should be only 17.
UDP source port	6	Type: Unsigned16 Valid value set: N/A Description: The source UDP port number of the TPDU transmitter.
UDP destination port	7	Type: Unsigned16 Valid value set: N/A Description: The destination UDP port number of the TPDU receiver.

### 7.3.3.3 Interface with the Transport Layer for outgoing protocol data units

The TL interaction with the security layer for an outgoing TPDU is summarized in Figure 42. When the security sub-layer receives the source address, source port, destination address, destination port, and payload size, it performs a lookup in the KeyDescriptor table to see whether security is enabled for that particular session. If the session's security level = 0 (NONE), a header size of 3 and a TMIC size of 0 shall be returned.. Otherwise the security sub-layer shall return the appropriate Crypto Key Identifier and TMIC sizes. All sessions at the transport layer are unicast; therefore, the Crypto Key Identifier length shall be either 0 or 1 depending on the number of valid keys available at that time for that security association.

The TL will then call the security sub-layer with the header and the payload. Depending on the security policy for that particular session, the payload may be encrypted, and a TMIC may be generated. The resulting header and payload will be returned to the transport layer for transmission.

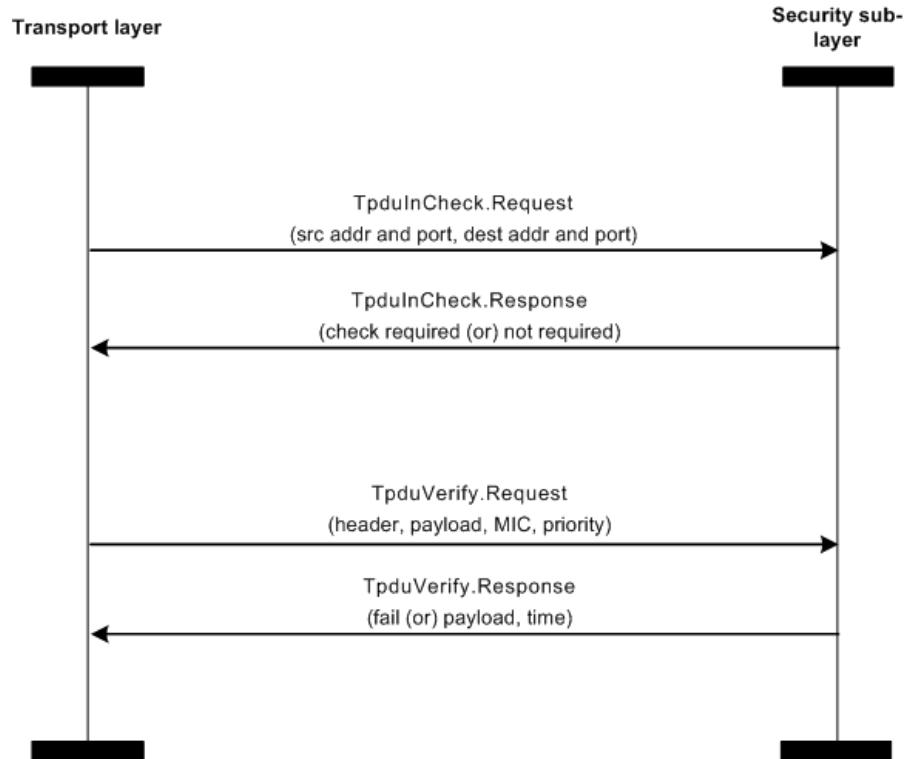


**Figure 42 – Transport layer and security sub-layer interaction, outgoing TPDU**

#### 7.3.3.4 Incoming protocol data unit processing overview

The transport layer interaction with the security layer for an incoming PDU is summarized in Figure 43. When the security sub-layer receives the source address, source port, destination address, and destination port, it performs a lookup in the MIB table to see whether security is enabled for that particular session and returns SEC\_CHECK\_REQUIRED or SEC\_CHECK\_NOT\_REQUIRED.

The transport layer shall then call the security sub-layer with the header, the payload, and the MIC. Depending on the security policy for that particular session, the payload may be decrypted, and a MIC may be verified. If the security check fails, a status of FAILURE will be returned, with the payload returned untouched. If the operation succeeds, the resulting payload and the recovered time of frame encoding will be returned to the transport layer.



**Figure 43 – Transport layer and security sub-layer interaction, incoming TPDU**

### 7.3.3.5 Transport layer interface to the security sub-layer

#### 7.3.3.5.1 General

The relationship between the TL and the security sub-layer is outlined in 7.3.3.2 for transport layer interface for an outgoing TPDU and 7.3.3.4 for transport layer interface for an incoming TPDU.

#### 7.3.3.5.2 Sec.TpduOutCheck.Request

##### 7.3.3.5.2.1 General

Sec.TpduOutCheck.Request is a check from the transport layer to the security sub-layer to obtain the size of the security fields (if any) required in the outgoing TPDU.

##### 7.3.3.5.2.2 Semantics of the service primitive

The semantics of Sec.TpduOutCheck.Request are as follows:

Sec.TpduOutCheck.Request (Source\_Address,  
Source\_Port,  
Destination\_Address,  
Destination\_Port,  
Payload\_Size )

Table 48 specifies the elements for the Sec.TpduOutCheck.Request.

**Table 48 – Sec.TpduOutCheck.Request elements**

Element name	Element identifier	Element scalar type
Source_Address	1	Type: Device address Valid value set: 128-bit address
Source_Port	2	Type: Integer Valid value set: 0-65535
Destination_Address	3	Type: Device address Valid value set: 128-bit address
Destination_Port	4	Type: Integer Valid value set: 0-65535
Payload_Size	5	Type: Integer Valid value set: 0-Assigned_Max_TSDU_Size; see 11.4.3.3

The security sub-layer shall use the Source\_Address, Source\_Port, Destination\_Address, and Destination\_Port to retrieve the appropriate policy (if any) for this security association.

#### 7.3.3.5.2.3 Appropriate usage

The TL invokes the Sec.TpduOutCheck.Request primitive to optionally protect a TPDU before it is transmitted.

#### 7.3.3.5.2.4 Effect on receipt

On receipt of the Sec.TpduOutCheck.Request primitive, the security sub-layer determines if the TPDU needs to be protected and returns the corresponding header and TMIC sizes.

#### 7.3.3.5.3 Sec.TpduOutCheck.Response

##### 7.3.3.5.3.1 General

Sec.TpduOutCheck.Response reports the result of a Sec.TpduOutCheck.Request.

##### 7.3.3.5.3.2 Semantics

The semantics of Sec.TpduOutCheck.Response are as follows:

```
Sec. TpduOutCheck.Response      (
    Sec_Header_Size,
    TMIC_Size
)
```

Table 49 specifies the elements for Sec.TpduOutCheck.Response.

**Table 49 – Sec.TpduOutCheck.Response elements**

Element name	Element identifier	Element scalar type
Sec_Header_Size (The additional header size required by the security sub-layer, in full octets)	1	Type: Integer Valid value set: 0-Assigned_Max_TSDU_Size; see Table 30
TMIC_Size (The size of the Transport Integrity Code, in full octets)	2	Type: Integer Valid value set: 0, 4, 8 or 16

##### 7.3.3.5.3.3 When generated

The security sub-layer generates Sec.TpduOutCheck.Response in response to a Sec.TpduOutCheck.Request. The Sec.TpduOutCheck.Response returns the additional sizes required to support the security layer functionality. A security association is a secure transport layer association based on:

- source address
- source port
- destination address

- destination port

#### **7.3.3.5.3.4 Appropriate usage**

On receipt of Sec.TpduOutCheck.Response, the transport layer is notified of the need to apply a security operation on the TPDU, along with the additional octets required to support that operation.

#### **7.3.3.5.4 Sec.TpduSecure.Request**

##### **7.3.3.5.4.1 General**

Sec.TpduSecure.Request instructs the security sub-layer to carry out the appropriate steps to secure an outgoing TPDU. The information about the security association is contained in the “pseudo-header” passed to the security sub-layer. See Figure 108 in subclause 11.4.2.

##### **7.3.3.5.4.2 Semantics of the service primitive**

The semantics of Sec.TpduSecure.Request are as follows:

```
Sec.TpduSecure.Request (TSS_Pseudo_Header,
                        TSS_Pseudo_Header_Size,
                        TSDU,
                        TSDU_Size)
```

Table 50 specifies the elements for the Sec.TpduSecure.Request.

**Table 50 – Sec.TpduSecure.Request elements**

Element name	Element identifier	Element scalar type
TSS_Pseudo_Header	1	Type: Data
TSS_Pseudo_Header_Size	2	Type: Integer Valid value set: 0-127
TSDU	3	Type: Data
TSDU_Size	4	Type: Integer Valid value set: 0- Assigned_Max_TSDU_Size; see Table 30

The security sub-layer shall obtain the source address, source port, destination address, and destination port from the TSS “pseudo-header”. That information is used to retrieve the appropriate keying material and policies for this security association.

The security sub-layer includes the Priority information in the TPDU header, protect the TPDU and generate the TMIC field as dictated by the PDU processing steps and current policies.

The security sub-layer populates the TL security header as specified in the TPDU processing steps and policies.

##### **7.3.3.5.4.3 Appropriate usage**

The TL invokes the Sec.TpduSecure.Request primitive to protect an outgoing TPDU after the TL receives the number of additional octets required in the transport header and the TMIC.

##### **7.3.3.5.4.4 Effect on receipt**

On receipt of the Sec.TpduSecure.Request primitive, the security sub-layer starts the appropriate PDU processing steps to protect the outgoing TPDU as dictated by the outgoing TPDU processing steps in 7.3.3.8.

#### **7.3.3.5.5 Sec.TpduSecure.Response**

##### **7.3.3.5.5.1 General**

Sec.TpduSecure.Response reports the result of a Sec.TpduSecure.Request.

##### **7.3.3.5.5.2 Semantics**

The semantics of Sec.TpduSecure.Response are as follows:

```
Sec.TpduSecure.Response
(
  TSS_Pseudo_Header,
  TSS_Pseudo_Header_Size,
  TSDU,
  TSDU_Size,
  TMIC,
  TMIC_Size
  Status
)
```

Table 51 specifies the elements for Sec.TpduSecure.Response.

**Table 51 – Sec.TpduSecure.Response elements**

Element name	Element identifier	Element scalar type
TSS_Pseudo_Header	1	Type: Data
TSS_Pseudo_Header_Size	2	Type: Integer Valid value set: 0-127
TSDU	3	Type: Data
TSDU_Size	4	Type: Integer Valid value set: 0- Assigned_Max_TSDU_Size; see Table 30
TMIC	5	Type: Data
TMIC_Size	6	Type: Integer Valid value set: 0, 4, 8, 16
Status	7	Type: Enumeration Valid value set: SUCCESS, FAILURE

### 7.3.3.5.5.3 When generated

The security sub-layer generates Sec.TpduSecure.Response in response to a Sec.TpduSecure.Request. The Sec.TpduSecure.Response returns populated security transport header, a possibly encrypted TSDU and a TMIC along with the appropriate sizes. Finally the Sec.TpduSecure.Response returns a status value that indicates either SUCCESS or the appropriate error code.

### 7.3.3.5.5.4 Appropriate usage

On receipt of Sec.TpduSecure.Response, the transport layer is notified of the result of protecting an outgoing TPDU.

### 7.3.3.5.6 Sec.TpduInCheck.Request

#### 7.3.3.5.6.1 General

Sec.TpduInCheck.Request instructs the security sub-layer to verify and possibly decrypt an incoming TL protocol data unit as appropriate.

#### 7.3.3.5.6.2 Semantics of the service primitive

The semantics of Sec.TpduInCheck.Request are as follows:

```
Sec.TpduInCheck.Request
(
  Source_Address,
  Source_Port,
  Destination_Address,
  Destination_Port,
  Payload_Size)
```

Table 52 specifies the elements for the Sec.TpduInCheck.Request.

**Table 52 – Sec.TpduInCheck.Request elements**

Element name	Element identifier	Element scalar type
Source_Address	1	Type: Device Address Valid value set: 128-bit address
Source_Port	2	Type: Integer Valid value set: 0-65535
Destination_Address	3	Type: Device Address Valid value set: 128-bit address
Destination_Port	4	Type: Integer Valid value set: 0-65535
Payload_Size	5	Type: Integer Valid value set: 0- Assigned_Max_TSDU_Size

The security sub-layer uses the Source\_Address, Source\_Port, Destination\_Address, and Destination\_Port to retrieve the appropriate policy (if any) for this security association.

#### 7.3.3.5.6.3 Appropriate usage

The TL invokes the Sec.TpduInCheck.Request primitive to check if a secure verification and possibly a decryption of a TPDU beforehand are required.

#### 7.3.3.5.6.4 Effect on receipt

On receipt of the Sec.TpduInCheck.Request primitive, the security sub-layer determines if the TPDU needs to be verified and optionally decrypted and returns a TRUE or FALSE.

#### 7.3.3.5.7 Sec.TpduInCheck.Response

##### 7.3.3.5.7.1 General

Sec.TpduInCheck.Response reports the result of a Sec.TpduInCheck.Request.

##### 7.3.3.5.7.2 Semantics

The semantics of Sec.TpduInCheck.Response are as follows:

```
Sec.TpduInCheck.Response      (
    Status
)
```

Table 53 specifies the elements for Sec.TpduInCheck.Response.

**Table 53 – Sec.TpduInCheck.Response elements**

Element name	Element identifier	Element scalar type
Status (the result of a Sec.TpduInCheck.Request primitive)	1	Type: Enumeration Valid value set: TRUE, FALSE

##### 7.3.3.5.7.3 When generated

The security sub-layer generates Sec.TpduInCheck.Response in response to a Sec.TpduInCheck.Request. The Sec.TpduInCheck.Response returns a status value that indicates either TRUE or FALSE depending on the policies on the current security association.

##### 7.3.3.5.7.4 Appropriate usage

On receipt of Sec.TpduInCheck.Response, the transport layer is notified of the need to call the Sec.TpduVerify.Request to verify and possibly decrypt the incoming TPDU.

#### 7.3.3.5.8 Sec.TpduVerify.Request

##### 7.3.3.5.8.1 General

Sec.TpduVerify.Request instructs the security sub-layer to verify an incoming TPDU and optionally decrypt the TSDU as appropriate.

### 7.3.3.5.8.2 Semantics of the service primitive

The semantics of Sec.TpduVerify.Request are as follows:

```
Sec.TpduVerify.Request (TSS_Pseudo_Header,
                      TSS_Pseudo_Header_Size,
                      TSDU,
                      TSDU_Size,
                      TMIC,
                      TMIC_Size)
```

Table 54 specifies the elements for the Sec.TpduVerify.Request.

**Table 54 – Sec.TpduVerify.Request elements**

Element name	Element identifier	Element scalar type
TSS_Pseudo_Header	1	Type: Data
TSS_Pseudo_Header_Size	2	Type: Integer Valid value set: 0-127
TSDU	3	Type: Data
TSDU_Size	4	Type: Integer Valid value set: 0- Assigned_Max_TSDU_Size; see Table 30
TMIC	5	Type: Data
TMIC_Size	6	Type: Integer Valid value set: 0, 4, 8 or 16.
Priority (4 bit)	7	Type: Integer Valid value set: 0-15

The security sub-layer verifies that the Transport Layer Security Header of the TPDU has the appropriate security control (octet 1) by comparing it to the current policy. The Crypto Key Identifier (octet 2), if present, is used to retrieve the correct key material. See 7.3.3.9.

The security sub-layer verifies the TMIC field as dictated by the current policies.

The time conveyed in the TL security header is used in the nonce construction for the authentication and optional decryption of the incoming TPDU.

The priority is provided to the security sub-layer in order to allow efficient implementation of replay protection.

### 7.3.3.5.8.3 Appropriate usage

The TL invokes the Sec.TpduVerify.Request primitive to verify and possibly decrypt a TPDU before it is transmitted.

### 7.3.3.5.8.4 Effect on receipt

On receipt of the Sec.TpduVerify.Request primitive, the security sub-layer starts the appropriate PDU processing steps to verify and optionally decrypt the incoming TPDU as dictated by the incoming PDU processing steps in 7.3.3.9.

### 7.3.3.5.9 Sec.TpduVerify.Response

#### 7.3.3.5.9.1 General

Sec.TpduVerify.Response reports the result of a Sec.TpduVerify.Request.

#### 7.3.3.5.9.2 Semantics

The semantics of Sec.TpduVerify.Response are as follows:

```
Sec.TpduVerify.Response
(
    TSDU,
    TSDU_Size,
    Time_Of_TPDU_Creation,
    Status
)
```

Table 55 specifies the elements for Sec.TpduVerify.Response.

**Table 55 – Sec.TpduVerify.Response elements**

Element name	Element identifier	Element scalar type
TSDU (after the option decryption)	1	Type: Data
TSDU_Size	2	Type: Integer Valid value set: 0- Assigned_Max_TSDU_Size; see 11.4.3.3
Time_Of_TPDU_Creation (32bit of time used in the nonce)	3	Type: UInt32 Valid value set: 0-2 <sup>32</sup> -1
Status	4	Type: Enumeration Valid value set: SUCCESS, FAILURE

#### 7.3.3.5.9.3 When generated

The security sub-layer generates Sec.TpduVerify.Response in response to a Sec.TpduVerify.Request. The Sec.TpduVerify.Response returns a status value that indicates either SUCCESS or the appropriate error code.

#### 7.3.3.5.9.4 Appropriate usage

On receipt of Sec.TpduVerify.Response, the transport layer is notified of the result of verifying and possibly decrypting the incoming TPDU.

#### 7.3.3.6 Transport protocol data unit security header structure

The TPDU security header structure is as described in Table 56.

**Table 56 – Structure of TL security header**

bits	7	6	5	4	3	2	1	0
octet								
1	Security_Control							
2 (opt)	Crypto_Key_Identifier							
3	Nominal_Time							
4	Nominal_Time							

The TL security header shall be added all TPDU to use header compression in NL. In case of no KeyDescriptor corresponding to certain TPDU, the TPDU shall be treated as no security (security level = NONE).

NOTE 1 If the TPDU has no TMIC, the UDP checksum is used for error detection.

Fields include:

- Security\_Control: As defined in 7.3.1.2.
- Crypto\_Key\_Identifier: Specifies the current Crypto Key Identifier used to protect this frame.
- Nominal\_Time: The time portion shall be 16 bits of truncated TAI time expressed in 2<sup>-10</sup> s in MSB coordination.

NOTE 2 Fixing the time granularity to the second gives the transport layer the ability to transmit 1023 TPDUs per second. With the maximum payload length of a TPDU, this is adequate for the throughput specified by 6LoWPAN. Variable granularity may be supported in future releases.

### 7.3.3.7 Nonce construction for transport protocol data units

This standard uses a different (but related) TPDU nonce construction than that of its DPDUs. A 13-octet nonce is required for the CCM\* engine. The nonce shall be constructed as the concatenation from first (leftmost) to last (rightmost) octets of data fields as shown in Table 57, wherein:

- The EUI-64 shall be used as an array of 8 octets and the truncated TAI time;
- The nominal TAI time of the TPDU creation shall be set at a granularity of  $2^{-10}$  s, and shall be no more than 1 second earlier than the actual local time of start of TPDU creation, and no more than 1 second later than the actual local time of end of TPDU creation. Each outgoing TPDU shall be created with a unique value for the 32-bit nominal TAI time for a given source EUI-64 and a given key. The maximum data rate of the TL shall be 1024 TDPUUs per second. The structure of the 32-bit nominal TAI time shall be as described in Table 58.

**Table 57 – Structure of the TPDU nonce**

octets	bits								
	7	6	5	4	3	2	1	0	
1	EUI-64								
...									
8									
9	Nominal TAI time of TPDU creation								
...									
12									
13	0xFF								

The 32-bit nominal TAI time is described in Table 58.

**Table 58 – Structure of 32-bit nominal TAI time**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Nominal TAI time (bits with weight $2^{21}$ to $2^{14}$ seconds)								
2	Nominal TAI time (bits with weight $2^{13}$ to $2^6$ seconds)								
3	Nominal TAI time (bits with weight $2^5$ to $2^2$ seconds)								
4	Nominal TAI time (bits with weight $2^{-2}$ to $2^{-10}$ seconds)								

At a  $2^{-10}$  s granularity, there will be 48.5 days before the 32-bit counter rolls over, thus providing at most 48.5 days before a new key needs to be deployed to avoid a potential nonce collision with resultant keystream reuse.

NOTE This representation is chosen because the sender and intended receivers are presumed to share the same time sense and the same nominal start time for a time slot that can be used for a MAC transaction.

### 7.3.3.8 Outgoing protocol data unit processing

The inputs to the TPDU security procedure are:

- The TPDU to be secured;
- The EUI-64 of the source device;
- The nominal TAI time;
- The source and destination 128-bit addresses; and
- The source and destination port.

The outputs from this procedure are:

- The status of the procedure; and

- If this status is SUCCESS, the secured frame.

The outgoing TPDU security procedure consists of the following steps:

- 1) The procedure shall obtain the KeyDescriptor from Table 94 meeting the following selection criteria:
  - The entries with Type = 10 (TL). If none are available, the procedure shall return with a status of UNAVAILABLE\_KEY.
  - Of those entries, the entries with the KeyLookupData matching the SourceAddress||SourcePort||DestinationAddress||DestinationPort (see Table 95) for this TPDU. If no KeyDescriptor is available and following two conditions are all true, the procedure shall treat the TPDU as no security (security level = NONE) TPDU. Otherwise, the procedure shall return with a status of UNAVAILABLE\_KEY.
  - Condition1: Join state of the receiving device is Provisioned or Joining (See Table 80).
  - Condition2: Source and destination ports are both for the DMAP (i.e., 0xF0B0).  
NOTE To transmit a join TPDUs that has security level = NONE, those conditions need to be satisfied.
  - Of those entries, the entries valid for the current period, satisfying the inequality ValidNotBefore < current time < ValidNotAfter shall be selected. If none are available, the procedure shall return with a status of UNAVAILABLE\_KEY.
  - Of those entries, if two or more keys are valid for the current time, the procedure shall select the key with the longest ValidNotAfter value.
  - Of those entries, if two or more keys have the same ValidNotAfter, the procedure shall select the key with the smallest ValidNotBefore.
  - Of those entries, if two or more keys have the same ValidNotBefore, the procedure shall select the key with the highest Crypto Key Identifier.
  - If that procedure fails, the procedure shall handle the TPDU as no security (security level = NONE).
- 2) The procedure shall retrieve the policy from selected KeyDescriptor
- 3) The procedure shall determine whether the TPDU to be secured satisfies the constraint on the maximum length of TPDUs, as follows:
  - The procedure shall set the length M, in octets, of the TMIC authentication field from the security level.
  - The length of the Key Index field in the TL security header shall be 1 octet, if more than 1 key is valid for the current security association and 0 otherwise.
  - The procedure shall determine the data expansion as Crypto Key Identifier length+M.
  - The procedure shall check whether the length of the TPDU to be secured, including data expansion, is less than or equal to the Assigned\_Max\_TSDU\_Size (see Table 30). If this check fails, the procedure shall return a status of TPDU\_TOO\_LONG.
- 4) The procedure shall build the security control octet of the TL security header. If the security level matches more than one KeyDescriptor from the current Key Descriptor, the Crypto Key Identifier shall be used with the Crypto Key Identifier Mode = 0x01; otherwise the procedure shall set the Crypto Key Identifier Mode = 0x00.
- 5) The procedure shall set the Crypto Key Identifier = Crypto Key Identifier from the current Key Descriptor (if present) in the TL security header. See Table 56.
- 6) The procedure shall build the 32-bit nominal TAI time as outlined in Table 58.
- 7) The procedure shall set the Nominal\_Time octets in the outgoing TL security header as the last 16 bits of the 32-bit nominal TAI time value. See octets 3 and 4 in Table 58.
- 8) If no Key Descriptor was found, then go to step 10; otherwise, the procedure shall use the 64-bit ID, the 32-bit nominal TAI time and the 8-bit value 0xFF to build the nonce as outlined in Table 57.

- 9) The procedure shall use the nonce, the key material, the TPDU header, the TPDU payload and the CCM\* mode of operation as described in IEEE Std 802.15.4-2006, 7.6.3.4, to secure the TPDU:
- If the SecurityLevel parameter specifies the use of encryption (see IEEE Std 802.15.4-2006, Table 97), the encryption operation shall be applied only to the TPDU's payload field. The corresponding payload field is passed to the CCM\* transformation process described in IEEE Std 802.15.4-2006, 7.6.3.4, as the unsecured payload. The resulting encrypted payload shall be substituted for the original payload.
  - The remaining fields in the TPDU, up to but not including the payload field, plus any required virtual fields, shall be passed to the CCM\* transformation process described in IEEE Std 802.15.4-2006, 7.6.3.4, as the non-payload field.
  - The ordering and exact manner of performing the encryption and integrity operations and the placement of the resulting encrypted data or integrity code within the TPDU payload field shall be as defined in IEEE Std 802.15.4-2006, 7.6.3.4.
- 10) The procedure shall return the secured frame and a status of SUCCESS.

### 7.3.3.9 Incoming protocol data unit processing

The input to the incoming TPDU security procedure is the TPDU to be unsecured, which contains the source and destination 128-bit addresses and the source and destination ports. The outputs from this procedure are the unsecured TPDU, the security level, the Crypto Key Identifier Mode, the key source, the key index, and the status of the procedure. All outputs of this procedure are assumed to be invalid unless and until explicitly set in this procedure. Each receiver of TPDUs maintains a cache of authenticated nonce values of recently received TPDUs.

The security procedure on TPDU reception consists of the following steps:

- 1) The procedure shall obtain the security level and the Crypto Key Identifier Mode from the corresponding subfields of the security control field and the key index from the corresponding subfields of the Crypto Key Identifier (if present) of the security header of the incoming TPDU.
  - 2) The procedure shall reconstruct the inferred sender's nominal TAI time of TPDU formation (see NOTE 2).
  - 3) The procedure shall compare the time in step 2 and the receiver's current TAI time. If the time in step 2 is more than 2 s ahead of the receiver's current TAI time, or more than N second behind the receiver's current TAI time (where N is an policy-determined parameter whose default value is 62 s) the security process returns FAILURE\_TPDU\_DID\_NOT\_AUTHENTICATE.
  - 4) The procedure shall obtain the KeyDescriptor from Table 94 meeting the following selection criteria.
    - The entries with Type = 10 (TL)
    - Of those entries, the entries with the KeyLookupData matching the SourceAddress||SourcePort||DestinationAddress||DestinationPort (see Table 95) for this TPDU. If no KeyDescriptor is available and following two conditions are all true, the procedure shall treat the TPDU as no security (security level = NONE) TPDU. Otherwise, the procedure shall return with a status of UNAVAILABLE\_KEY.
      - Condition1: Join state of the receiving device is Provisioned or Joining (See Table 80).
      - Condition2: Source and destination ports are for both DMAP's (i.e. 0xF0B0).
- NOTE 1 To receive a join TPDUs that has security level = NONE, this judgment is needed.
- Of those entries, the entries valid for the current period, satisfying the inequality ValidNotBefore < current time < ValidNotAfter shall be selected. If none is available, the procedure shall return with a status of UNAVAILABLE\_KEY.
  - Of those entries, if two or more keys are valid for the current time, the procedure shall select the key with the longest ValidNotAfter value.

- Of those entries, if two or more keys have the same ValidNotAfter, the procedure shall select the key with the smallest ValidNotBefore.
  - Of those entries, if two or more keys have the same ValidNotBefore, the procedure shall select the key with the highest Crypto Key Identifier.
  - If that procedure fails, the procedure shall return with a status of UNAVAILABLE\_KEY.
- 5) The procedure shall determine whether the security level of the incoming TPDU conforms to the security level policy by comparing the SecurityLevel of the matching Key Descriptor obtained from step 2 above. If there is a mismatch, the procedure shall return with a status of IMPROPER\_SECURITY\_LEVEL.
- 6) The procedure shall then use the EUI-64 of the sender, the nominal TAI time of TPDU formation from step 2, the received lower 16 bits of nominal TAI time (see Table 58) to generate the nonce outlined in Table 57.
- 7) The procedure shall use the nonce, the key from the Key Descriptor obtained in step 2, the headers (the non-payload fields), the payload and the MIC of the incoming TPDU and the CCM\* mode of operation as described in operations (see IEEE Std 802.15.4-2006, 7.6.3.5) to authenticate and optionally decrypt the TPDU:
- If the security level specifies the use of encryption (see IEEE Std 802.15.4-2006, Table 96), the decryption operation shall be applied only to the actual TPDU payload field (see IEEE Std 802.15.4-2006, 7.2.2.2.2). The corresponding payload field shall be passed to the CCM\* inverse transformation process described in IEEE Std 802.15.4-2006, 7.6.3.5 as the secure payload.
  - The remaining fields in the TPDU, plus any required virtual fields, shall be passed to the CCM\* inverse transformation process described in IEEE Std 802.15.4-2006, 7.6.3.5 as the non-payload fields (see IEEE Std 802.15.4-2006, Table 100).
  - The ordering and exact manner of performing the decryption and integrity checking operations and the placement of the resulting decrypted data within the TPDU payload field shall be as defined in IEEE Std 802.15.4-2006, 7.6.3.5.
- 8) If the CCM\* inverse transformation process fails, then the procedure may decrement the nominal TAI time by 64 s and repeat the above process during DSMO.pduMaxAge period. Otherwise, the procedure shall set the TPDU to be unsecured and return a status of SECURITY\_ERROR.
- 9) The procedure shall lookup the nonce of the just authenticated TPDU is in the cache, with the following possible outcomes:
- The time of the nonce is older than the oldest time in the cache and the cache does not have space for an additional older entry, so the security process returns FAILURE\_OVERAGE\_TPDU.
  - The nonce is already in the cache, so the security process returns FAILURE\_DUPLICATE\_TPDU.
  - Any encrypted payload of the TPDU is decrypted, and the security process returns SUCCESS.
- 10) The procedure shall insert the nonce value in the cache, if necessary bumping from the cache the cache entry with the oldest inferred nominal TAI time of TPDU formation and return with the unsecured TPDU, the security level, the Crypto Key Identifier Mode, the key source, the key index (if present) and a status of SUCCESS.

NOTE 2 The sender's nominal TAI time of TPDU formation is inferred initially from the receiver's current TAI time and the fractional nominal TAI time of TPDU creation that is specified in the TPDU, using the relationship (current-receiver-time + 2 s) >= sender's-nominal-time >= (current-receiver-time - DSMO.pduMaxAge)), where the implementation can choose which of the two limiting equalities it wishes to honor. The time duration of 2 sec is intended cover +/- 1 plus edge condition.

NOTE 3 It is permitted for the cache to be segmented into separate caches for each sending EUI-64 address. It is further permitted for the cache to be segmented by the reported network-layer QoS, so that a cache that holds only a few nonces of low-priority TPDUs need not also hold dozens to hundreds of nonces for overtaking higher-priority TPDUs. It is further permitted for the cache size to be adaptive, so that repeated occurrences of the first outcome in step 9 cause the cache to grow, with appropriate reduction in cache size if and when the excess cache capacity has not been used for an extended period of time.

### **7.3.3.10 Detection and discard of duplicated or replayed transport layer protocol data units**

See step 9 above.

## **7.4 The join process**

### **7.4.1 General**

The join process describes the steps by which a new device is admitted into a standard-compliant network and obtains all the relevant information to be able to communicate with other devices as well as the system manager and security manager.

NOTE This section assumes a device that has an ISA100.11a data link layer protocol stack for completeness. Since it is an application layer protocol, it will also work for devices that do not have a data link layer by omitting the data link layer steps.

### **7.4.2 Prerequisites**

The join process follows the provisioning step, during which cryptographic information and non-cryptographic configuration parameters may be provided to the new device. A new device shall obtain such necessary provisioning information from the provisioning device. This is described in Clause 14. The Join\_Command attribute in the DMO of a device shall be used to command the device to join the network.

A joining device shall join the target network with one of the following security options:

- Symmetric keys
- Asymmetric keys
- No security

The no security option uses any secret keys for transfer of join keys. Instead, it uses Global key (i.e., K\_global or K\_open) as defined in 3.3.1.3. The MIC then is the equivalent of a strong CRC with no security guarantees, but with a very high probability of detection of random errors. Additionally, no end-to-end secure sessions are allowed.

A device implementing the symmetric key join process option shall have the following security-related information:

- A 128-bit join key.
- The 64-bit unique ID of the security manager that shares the join key.

A device implementing the asymmetric-key join process option shall have the following security-related information:

- Certificate signed by a certificate authority trusted by the target network.

A device implementing the no security join process option shall have the following security-related information:

- The well-known, published, non-secret 128-bit key common to all standard-compliant networks, K\_global as defined in 3.3.1.3.

### **7.4.3 Desired device end state and properties**

At the conclusion of the join process, the system shall have the following state:

- The new device and the security manager shall securely share a symmetric long-term master key;
- If a DL is present, the new device shall have the required cryptographic material at the DL to talk to its direct neighbors;
- If a DL is present, the new device shall have the required non-cryptographic material and resources at the DL to talk to at least one of its direct neighbors; and
- The new device shall have a contract with the system manager.

NOTE 1 A contract with the system manager includes a transport session key shared between the system manager and the new device.

When using either the secure symmetric key or asymmetric key option, the join process provides the following security assurances:

- Protection against replay attacks on join APDUs.
- Cryptographic assurance to the new device that the security manager is alive.
- Cryptographic assurance to the security manager that the new device is alive.
- Authenticity.
- Cryptographic assurance that the join request comes from a device that has valid trust material.
- Cryptographic assurance that the join APDUs have not been altered.
- Secrecy.
- The keys in the join reply shall be cryptographically protected, such that an eavesdropper cannot recover the transported keys.

NOTE 2 A challenge / response protocol is used for the secure join process to eliminate the reliance on a trusted time source at time of join.

#### **7.4.4 Join process steps common for symmetric key and asymmetric key options**

##### **7.4.4.1 General**

When using the secure symmetric key option or asymmetric key option for the join process, the device goes through the following general steps to complete the join process.

The system manager controls the process of a new device joining the network. A non-joined device that implements a DL as per this standard, listens for advertisement DPDUs from local routers whose advertisement functions are configured by the system manager.

Advertisements can be discovered by using active scanning, passive scanning, or a combination of both. Active scanning involves solicitation DPDUs from the joining device to trigger the transmission of advertisements on request. Detailed information for active/passive scanning is found in 9.1.13.

Such an advertising router shall assist during the join process of a new device by acting as a proxy for the system manager. This advertising router forwards the join request from the new device to the system manager and forwards the join response from the system manager to the new device. A join request from the new device shall be processed by the system manager. The system manager shall generate a join response after communicating with the security manager.

##### **7.4.4.2 Construction of join process messages**

The join request from a new device is divided into separate messages to separate out the security information, exchanged between the device and the security manager, from the non-security information, exchanged between the device and the system manager. The join response is also similarly divided into separate messages to separate out the security information from the non-security information.

The non-security information exchanged during the join process is described in 6.3.9.2. This is done using the method defined in 6.3.9.2.2 for the advertising router's DMO and methods defined in 6.3.9.5 for the system manager's DMSO.

The security information exchanged during the join process is dependent on the join option used. The symmetric key option is described in 7.4.5 and the asymmetric key option is described in 7.4.6.

In order for the new device to construct the join process messages and send them to the advertising router, it needs to know the EUI-64 of the advertising router. The process followed by the new device to obtain this EUI-64 of the advertising router is described in 9.1.14. Using this EUI-64 of the advertising router and the EUI-64 of the new device, the join process request messages from the new device to the advertising router as well as the join process

response messages from the advertising router to the new device shall be constructed as follows:

- If there is a DL present, the DL header for these join process messages shall be constructed as described in 9.3.
- The NL header for these join process messages shall be constructed as described in 10.5.3.
- The TL header for these join process messages shall be constructed as described in 11.5. For calculation of the UDP checksum, the UDP pseudo header for IPv6 shall use the link local IPv6 addresses of the new device and the advertising router as the source and destination addresses. These link local IPv6 addresses shall be computed using the EUI-64 of the new device and the EUI-64 of the advertising router.
- When the DMAP in the new device sends the join process request messages down to the TL, it shall use the 128-bit link local address of the new device and the 128-bit link local address of the advertising router as the source address and the destination address respectively. When the DMAP in the advertising router sends the join process response messages down to the TL, it shall use the 128-bit link local address of the advertising router and the 128-bit link local address of the new device as the source address and the destination address respectively.
- At the TL, the Sec.TpduOutCheck.Response shall return with a value of 3 for the security header size and 0 for the TMIC size, indicating a TL security header with security level = 0 (NONE) and no security protection is necessary on the outgoing join requests from the new device and the outgoing join responses from the advertising router.
- If there is a DL present, at the DL, the security header which has security level = 1 (MIC-32) and 32-bit DMIC for these join process messages shall be constructed as described in 7.3.2.5 using K\_global (Crypto Key Identifier = 0). This 32-bit DMIC is used only to detect random errors in the outgoing join requests from the new device and the outgoing join responses from the advertising router. The advertising router shall use its existing contract with the system manager to forward the join process messages on behalf of the new device.

NOTE 1 A new device that is trying to join the network does not have any contracts assigned by the system manager. So, this communication between the new device and the advertising router is not based on any contract.

NOTE 2 The PSMO.Security\_Sym\_Confirm() request and response messages should be protected by the DL and TL key information that distributed in the PSMO.Proxy\_Security\_Sym\_Join() response message.

#### **7.4.4.3 Protection of join process messages**

##### **7.4.4.3.1 General**

As the new device does not have the necessary DL subnet key and a TL level session key with the advertising router, all join process messages, other than confirm messages, between the new device and the advertising router shall use the K\_global at the DL level to construct a 32-bit DMIC. At the TL level, the UDP checksum shall be used for these messages.

The security information in the join request, as well as all the information coming back from the system manager and the security manager in the join response messages, is protected using the join key. This is described in 7.4.5 for the symmetric key option and in 7.4.6 for the asymmetric key option.

##### **7.4.4.3.2 Join PDU replay attack prevention (INFORMATIVE)**

To protect against a join PDU replay attack, it is recommended that the security manager check for duplicate challenges with a valid MIC from the new device. If no challenge duplicates are detected, the security manager stores the challenge value for further duplication checking. In the event of a duplicate detection, the security manager discards the PDU before processing the join PDU.

##### **7.4.4.3.3 Protecting non-security message in the join process**

###### **7.4.4.3.3.1 General**

This section describes the configuration of the security settings during the Join Process. The non-security related network information is configured with the

DMSO.System\_Manager\_Join() and DMSO.System\_Manager\_Contract() methods. The details of the methods are specified in 6.3.9.2. Such non-security messages are protected with cryptographic operations at the Application Layer. The non-security messages are generated in the system manager and passed to the security manager which then adds the MIC using the join key. The protected messages are transmitted to the joining device via the DMSO in the system manager. At the joining device, the MIC in the received response is validated with the same operation.

At the joining device, the MIC in the received response is validated with same operation.

#### **7.4.4.3.3.2 MIC generation**

##### **7.4.4.3.3.2.1 System\_Manager\_Join response**

The DMSO.System\_Manager\_Join() method is defined in 6.3.9.5. The MIC field is the most significant 4 octets in MACTag generated with the following operation.

$$\text{MACTag} = \text{HMAC- MMOK\_join}[\text{Output Argument \#1 to \#6 in Table 23} \parallel \text{EUI-64}_{\text{join\_device}} \parallel \text{Challenge}_{\text{join\_device}}]$$

##### **7.4.4.3.3.2.2 System\_Manager\_Contract response**

The DMSO.System\_Manager\_Contract method is defined in 6.3.9.5. The MIC field is the most significant 4 octets in MACTag generated with following operation.

$$\text{MACTag} = \text{HMAC- MMOK\_join}[\text{Output Argument \#1 in Table 24} \parallel \text{EUI-64}_{\text{join\_device}} \parallel \text{Challenge}_{\text{join\_device}}]$$

#### **7.4.4.3.3 Confirmation**

After DMO.Proxy\_System\_Manager\_Join().Response and DMO.Proxy\_System\_Manager\_Contract().Response are received, the join device sends a message to inform that the correct network information has been received by the system manager.

In the symmetric key join process, the confirmation process is integrated in the PSMO.Security\_Confirm() method specified in Table 61.

In asymmetric key join process, the confirmation process is accomplished with PSMO.Network\_Information\_Confirmation() method specified in Table 75.

#### **7.4.4.4 Join timers**

In the join process, 2 timers are defined. Upon expiration of either of those timers, any information (e.g., state and received parameter) cached for particular join process shall be removed or re-initialized.

T<sub>1</sub>: Time duration managed in the joining device, from the time of transmission of the security join request, to the time of correct validation of the confirmation response generated by the security manager. If the joining device is not a backbone device, the actual value for T<sub>1</sub> shall be set a DauxJoinTimeout value that distributed within DL advertisement (see Table 127). Otherwise, the actual value for T<sub>1</sub> shall be set to 60 sec for the backbone device.

T<sub>2</sub>: Time duration managed in the security manager, from the time of reception of the security join request to the time of correct validation of the security confirmation message generated by the joining device. The actual value of T<sub>2</sub> is not specified in this specification.

NOTE It is recommended that T<sub>2</sub> be less than T<sub>1</sub>.

#### **7.4.4.5 Join process of backbone device**

A backbone device joins a target network by executing the join method in the system manager's DMO instead of the advertisement router's. Therefore, the backbone device does not need to discover an advertisement router; the DPO.Target\_System\_Manager\_Address

shall be set in provisioning phase. The overview of the backbone device join process is illustrated in Figure 45 for the symmetric key join process and Figure 48 for the asymmetric key join process.

#### 7.4.4.6 TMIC length constraints for session between join node and system manager

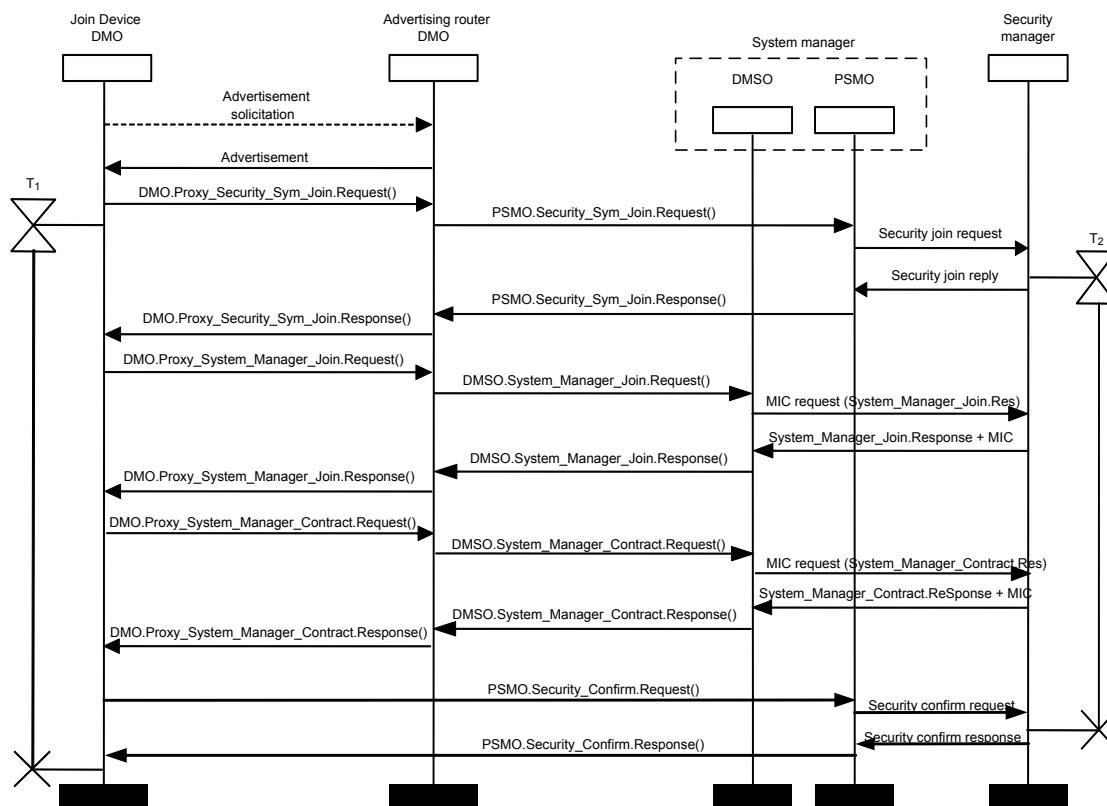
At the end of join process, the security manager assigns an initial security level for the session between the joining device and the system manager. The security level for the session shall not have 128-bit TMIC (i.e., security level, 0x03 for MIC-128 and 0x07 for ENC-MIC-128) during system operation.

#### 7.4.5 Symmetric key join process

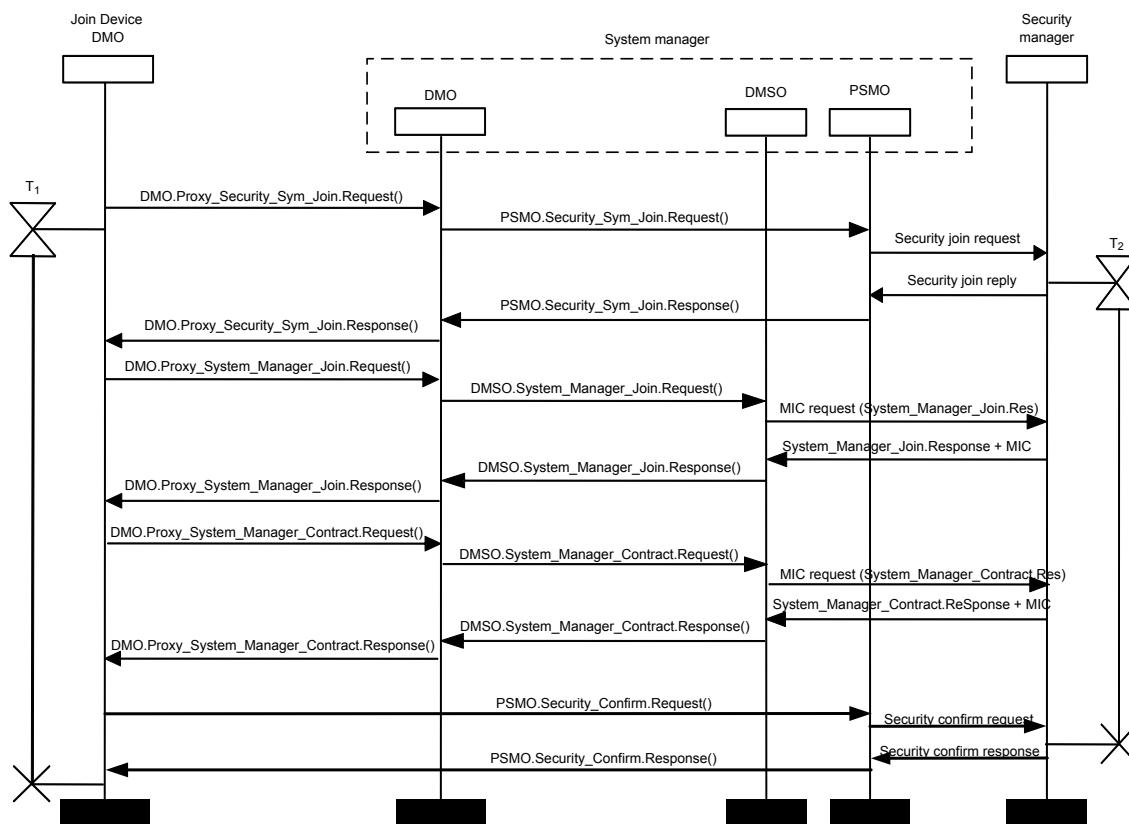
##### 7.4.5.1 General

Figure 44 illustrates the messaging involved in the symmetric key join process by which a new device shall join an operating network in which it has not recently been a participant. The flow shows the normal case in which no errors or timeouts occur. The timeouts are specified in Table 93.

On the joining device, the symmetric key join process shall be initiated with a DMO.Proxy\_Security\_Sym\_Join().Request and finalized with a valid PSMO.Security\_Confirm().Response. On the security manager, the symmetric key join process shall be initiated with a valid message derived from PSMO.Security\_Sym\_Join().Request and finalized with a valid message derived with PSMO.Security\_Confirm().Request.



**Figure 44 – Example: Overview of the symmetric key join process**



**Figure 45 – Example: Overview of the symmetric key join process of backbone device**

As shown in Figure 44, a new device shall use the methods defined for the advertising router's DMO to send and receive the join request and join response messages. The methods related to the non-security information are described in 6.3.9.2. The DMO methods related to the security information are described in 7.4.5.2. After transmission of the DMO.Proxy\_Security\_Sym\_Join().Request, the new device shall start to count timer T<sub>1</sub>.

The advertising router shall use the methods defined for the system manager's DMSO to send and receive the non-security related join request and response messages. These DMSO methods are described in 6.3.9.5. Methods defined for the system manager's proxy security management object (PSMO) shall be used by the advertising router to send and receive the security related join request and response messages. The PSMO methods related to the security information are described in 7.4.5.2. As shown in Figure 44, the PSMO receives the security-related join request and forwards it to the security manager. The security manager may check a white or black list for the device and ask for human verification before deciding to admit or reject the joining device. If the result is positive, the security manager verifies the cryptographic information of the join request. If the checks fail, the system manager is instructed to revoke the resources allocated to the new device. If the test succeeds, the security manager does the following:

NOTE 1 The methodology of filtering the joining device in security manager is beyond the scope of this specification.

- 1) Starts to count join timer T<sub>2</sub>
- 2) Generates a new master key for the new device.
- 3) Creates a new secure session for the contract between the system manager and the new device.
- 4) Retrieves the current DL key and Crypto Key Identifier for the new device's subnet.
- 5) Generates a fresh, unique challenge for the new device.

- 6) Cryptographically protects the aforementioned keys and forms a message integrity check code on the entire response.
- 7) Sends the security-related response, including the message integrity check code, back to the PSMO.

The PSMO sends this security-related response back to the advertising router which in turn forwards it to the new device.

The new device checks the cryptographic integrity of this security-related response. If the test fails, the received response is dropped. If the test succeeds, then the security-related response is processed by the device and stops to count join timer T<sub>1</sub>.

The non-security related join response that is generated by the system managers DMSO is forwarded to the security manager in order to cryptographically protect the information in this response. Once the DMSO receives this protected response, it sends this protected response back to the advertising router which in turn forwards it to the new device. The new device checks the cryptographic integrity of this protected response before using the information in this response to complete the join process. This response includes information about a contract that the system manager established between the new device and the system manager. This contract is described in 6.3.11.2.6.6.

As part of the last step of the join process, the new device shall send back a security confirmation to the security manager which consists of the challenge from the security manager, authenticated under the new master key. This security confirmation is sent to the PSMO which forwards it to the security manager. The PSMO method used for this is described in 7.4.5.2.

The security manager checks the confirmation message. If the test fails, the received confirmation response, the join state and the cached information for the new device shall be dropped. If the test succeeds, then the security manager stops to count T<sub>2</sub>, and sends a confirmation response back to the new device.

If the new device receives a positive response against its confirmation request, then the new device stops to count T<sub>1</sub>.

The contract that was established between the new device and the system manager during the join process is used to support these messages.

NOTE 2 By sending the response of the challenge authenticated under the master key, the new device proves to the security manager that it was able to retrieve the master key, and therefore that it had the join key.

NOTE 3 Multiple method calls may be concatenated into a single TSDU as determined by the ASL. For example, to reduce the traffic overhead or the join time, the Proxy\_Security\_Sym\_Join().Request and the Proxy\_System\_Manager\_Contract().Request may be concatenated in the same TSDU sent to the advertising router's DMO

#### **7.4.5.2 Device management object and proxy service management object methods related to the symmetric key join process**

##### **7.4.5.2.1 General**

The new device shall use the Proxy\_Security\_Sym\_Join method defined for the advertising routers DMO in the advertising router to send its security information that is part of the join request and to get its security information that is part of the join response.

NOTE 1 To mitigate a join message flooding, system manager limit wireless resources (e.g., timeslots) assigned in advertisement routers for receiving joining messages. The resources are described in 9.3.5.2.4.2.

Table 59 describes the Proxy\_Security\_Sym\_Join method. The source object for invoking the DMO.Proxy\_Security\_Sym\_Join().Request shall be the DMO in the Joining device's DMAP.

**Table 59 – Proxy\_Security\_Sym\_Join method**

<b>Standard object type name: DMO (Device management object)</b>			
<b>Standard object type identifier: 127</b>			
<b>Defining organization: ISA</b>			
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>	
Proxy_Security_Sym_Join	5	Method to use advertising router as proxy to send security join request and get security join response	
<b>Input arguments</b>			
<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>	<b>Argument description</b>
1	Join_Request	Data type: Security_Sym_Join_Request; see 7.4.5.2.2	Security join request based on symmetric keys from new device that needs to be forwarded to security manager.
<b>Output arguments</b>			
<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>	<b>Argument description</b>
1	Join_Response	Data type: Security_Sym_Join_Response; see 7.4.5.2.3	Security join response based on symmetric keys from security manager that needs to be forwarded to new device; this is protected using the join key.

The advertising router shall use the Security\_Sym\_Join method defined for the system manager's PSMO for sending the security information that is part of the join request on behalf of the new device and to get the security information that is part of the join response.

Table 60 describes the Security\_Sym\_Join method.

**Table 60 – Security\_Sym\_Join method**

<b>Standard object type name: PSMO (Proxy security management object)</b>			
<b>Standard object type identifier: 105</b>			
<b>Defining organization: ISA</b>			
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>	
Security_Sym_Join	1	Method to use the PSMO in the system manager to send security join request and get a security join response	
<b>Input arguments</b>			
<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>	<b>Argument description</b>
1	Join_Request	Data type: Security_Sym_Join_Request; see 7.4.5.2.2	Security join request from new device to security manager.
<b>Output arguments</b>			
<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>	<b>Argument description</b>
1	Join_Response	Data type: Security_Sym_Join_Response; see 7.4.5.2.3	Security join response from security manager to new device that is protected using the join key

As part of the last step of the join process, the new device shall use the Security\_Confirm method defined for the system manager's PSMO for sending a security confirmation to the security manager.

Table 61 describes the Security\_Confirm method.

**Table 61 – Security\_Confirm method**

<b>Standard object type name: PSMO (Proxy security management object)</b>			
<b>Standard object type identifier: 105</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Security_Confirm	2	Method used by new device to send security confirmation to the security manager through the PSMO	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Security_Sym_Confirm	Data type: Security_Sym_Confirm; see 7.4.5.2.4	Security confirmation from new device to security manager.
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
-	-	-	-

NOTE 2 Although the Security\_Confirm method does not have any output arguments, the Execution response message in Application Sub-Layer is returned as result of this method.

#### 7.4.5.2.2 Symmetric key join request

The Security\_Sym\_Join\_Request data structure that is used to form the symmetric key join request is defined in Table 62.

**Table 62 – Security\_Sym\_Join\_Request data structure**

<b>Standard data type name: Security_Sym_Join_Request</b>		
<b>Standard data type code: 410</b>		
Element name	Element identifier	Element type
New_Device_EUI64	1	Type: Unsigned64 Classification: Constant Accessibility: Read only Default value : N/A Valid value set: N/A
128_Bit_Challenge_From_New_Device	2	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Algorithm_Identifier	3	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : 1 Valid value set: N/A
MIC	4	Type: Unsigned32 Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A

Fields include:

- New\_Device\_EUI64 is the EUI-64 of the joining device. This EUI-64 is used by the advertising router when forwarding the message to the system manager to identify this device uniquely, as there could be multiple new devices joining at the same time
- 128\_Bit\_Challenge\_from\_new\_device is a fresh unique challenge generated by the new device to verify that the security manager is alive.

- The algorithm identifier shall be used to specify the symmetric key algorithm used in the target network. The value of 0x0 shall be reserved. A symmetric key algorithm of 0x01 corresponding to AES\_CCM\* shall be the only symmetric algorithm and mode supported for the join process.

NOTE Currently, only AES\_CCM\* is defined as a symmetric key algorithm. However, this field is prepared for algorithms for future use or national regulation.

- The MIC is of 32 bits in length and is computed over the elements 1 through 4, using the join key and the 13 most significant octets of the challenge as nonce.

#### 7.4.5.2.3 Symmetric key join response

The Security\_Sym\_Join\_Response data structure that is used to form the symmetric key join response is defined in Table 63.

**Table 63 – Security\_Sym\_Join\_Response data structure**

<b>Standard data type name: Security_Sym_Join_Response</b>		
<b>Standard data type code: 411</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
128_Bit_Challenge_From_SecurityManager	1	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
128_Bit_Response_To_New_Device_Hash_B	2	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Combined_Security_Level	3	Type: Unsigned8 (see Table 64) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Master_Key_HardLifeSpan	4	Type: Unsigned16 (See Table 66) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
DL_Key_HardLifeSpan	5	Type: Unsigned16 (See Table 66) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Sys_Mgr_Session_Key_HardLifeSpan	6	Type: Unsigned16 (See Table 66) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
DL_Key_ID	7	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Encrypted_DL_Key	8	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Encrypted_Sys_Mgr_Session_Key	9	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A

This data structure consists of a plaintext section and an encrypted section. The plaintext section shall be composed of the header, the original challenge from the new device, and a new challenge from the security manager which is different from the challenge generated by the new device, and the key policies. The encrypted section shall be composed of the DL key and the session key with the system manager.

- 128\_Bit\_Challenge\_From\_Security\_Manager is a fresh unique challenge generated by the Security Manager to verify that the new device is alive
- 128\_Bit\_Response\_To\_New\_Device\_Hash\_B shall be calculated as:

$$\text{Hash\_B} = \text{HMAC-MMOK}_{\text{join}}[\text{challenge\_from\_security\_manager} \parallel \text{challenge\_from\_new\_device} \parallel \text{EUI-64new\_device} \parallel \text{EUI-64security\_manager} \parallel \text{Message\_Key\_Transport}]$$

$$\text{Message\_Key\_Transport} = \text{Combined\_Security\_Level} \parallel \text{Master\_Key\_HardLifeSpan} \parallel \text{DL\_Key\_HardLifeSpan} \parallel \text{Sys\_Mgr\_Session\_Key\_HardLifeTime} \parallel \text{DL\_Key\_ID} \parallel \text{Encrypted DL key} \parallel \text{Encrypted SysMan session key}$$

- The Master\_Key\_HardLifeSpan, DL\_Key\_HardLifeSpan and Sys\_Mgr\_Session\_Key\_HardLifeSpan shall be the 16 bit HardLifeSpan in hours. The Key Type='001' and Key Usage can be inferred implicitly from Table 90 and Table 91 by the element Identifier. A default granularity of 0x2='hours' shall be used for the policies in the join response message.
- The DL\_Key\_ID shall be the Crypto Key Identifier associated with the DL Key sent in the join response. The Crypto Key Identifier of the master key and session key shall be set implicitly (not transmitted but inferred) as 0x00.
- The 13 most significant octets of the challenge sent from security manager shall be used as the nonce to encrypt DL key and session key. The DL key and Session Key are encrypted in the same time (single operation of AES-CCM\* encryption with MIC length = 0).
- Response to new device shall be the keyed hash defined as follows:
- The first master key shall be derived as:

$$K_{\text{master}} = \text{HMAC-MMOK}_{\text{join}}[\text{EUI-64new\_device} \parallel \text{EUI-64security\_manager} \parallel \text{challenge\_from\_new\_device} \parallel \text{challenge\_from\_security\_manager}]$$

- The DL key and the session key to support the contract with the system manager shall be encrypted using the master key.

NOTE 1 By including the challenge from the new device and calculating a MIC over it, the security manager proves that it is a live device with knowledge of the join key.

NOTE 2 16 bits of validity period in hours gives a range of over 7 years which is adequate to express the current maximum key lifetime.

**Table 64 – Structure of compressed Security level field**

Octets	Bits							
	7	6	5	4	3	2	1	0
1	DL Security Level			Sys Mgr Ses Security Level			Master key Sec level	

Fields include:

- DL\_Security\_Level: The security level applied to the DL key conveyed in the Security\_Sym\_Join response message. The format of this field shall be as follows in Table 35.
- Sys\_Mgr\_Ses\_Security\_Level: The security level applied to the session key with the system manager conveyed in the Security\_Sym\_Join response message. The format of this field shall be as follows in Table 35.
- Master\_Key\_Sec\_Level: The MIC length applied to the Master key generated in the join process. The format of this field is defined in Table 66. Since the encryption factor is different in each message protected by the Master key, only the MIC length is specified in this field. The actual security level shall be selected from Table 65 with a combination of encryption conditions in each message.

NOTE 3 For example, since the Security\_New\_Session\_Request and the Security\_New\_Session\_Response data structure don't have any elements to be encrypted, the Security\_Level in the Security\_Control field is set to MIC-n with the MIC length specified in this structure. While the Security\_Key\_and\_Policies data structure has element to be encrypted, the Security\_Level in the Security\_Control field is set to ENC-MIC-n with MIC length specified in this structure.

**Table 65 – Master key security level**

Security level identifier	Master_Key_Sec_Level field $b_1b_0$	Security attributes
0x00	00	Reserved
0x01	01	MIC-32
0x02	10	MIC-64
0x03	11	MIC-128

NOTE 4 Since a MIC is always used to protect the join process PDUs with the Master key, the security level identifier 0x00 is Reserved.

**Table 66 – Structure of KeyHardLifeSpan field**

Octets	Bits							
	7	6	5	4	3	2	1	0
1	HardLifeSpan							
2	HardLifeSpan							

Fields include:

- HardLifeSpan (hours): The HardLifeSpan of the key validity period in hours in MSB coordination.

Inferred parameters regarding key lifetime are generated as following;

- ValidNotAfter = Absolute TAI time when APDU is received + Nominal HardLifeSpan in Hours
- SoftExpirationTime = Absolute TAI time when APDU is received + (Nominal HardLifeSpan in Hours / 2)
- SoftLifeSpan = HardLifeSpan in Hours / 2. If the HardLifeSpan is infinite (0x0000), the SoftLifeSpan shall be set to infinite (0x0000).
- ValidNotBefore = The TAI time when APDU is received

A SoftLifeSpan of 50% of the HardLifeSpan shall be used as a default for keys sent with a KeyHardLifeSpan field.

The ValidNotBefore can be inferred as the reconstructed time used in the authentication of the PDU, and is not included due to space restrictions in the response PDU.

The time that the APDU is received shall not be more than DSMO.pduMaxAge seconds after it was created. That gives an acceptable start time.

#### 7.4.5.2.4 Symmetric key security confirmation

The Security\_Sym\_Confirm data structure that is used to form the symmetric key security confirmation is defined in Table 67. The source object for invoking PSMO.Security\_Sym\_Confirm().Request shall be DMO in joining device's DMAP.

**Table 67 – Security\_Sym\_Confirm data structure**

<b>Standard data type name: Security_Sym_Confirm</b>		
<b>Standard data type code: 412</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
128_Bit_Response_To_Security_Manager	1	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A

128\_Bit\_Response\_To\_Security\_Manager shall be calculated as:

HMAC-MMOK\_join[challenge\_from\_new\_device || challenge\_from\_security\_manager || EUI-64new\_device || EUI-64security\_manager || MIC<sub>1</sub> || MIC<sub>2</sub>]

Where: MIC<sub>1</sub> – 32-bit MIC value in System\_Manager\_Join response and MIC<sub>2</sub> – 32-bit MIC value in System\_Manager\_Contract response.

NOTE 1 The join confirmation tells the security manager that the device was able to recover the master key using the join key, thus providing proof that the device, which knows the join key, is alive.

NOTE 2 The construction of the hash for the challenge-response protocol was modeled after the protocol outlined in 10.17 of the Handbook of Applied Cryptography, A. Menezes, P. van Oorschot, and S. Vanstone, CRC Press, 1996.

#### **7.4.6 Asymmetric key join process**

##### **7.4.6.1 Overview**

The asymmetric key join process, like the symmetric key join process specifies the sequence of steps by which and the conditions under which a device may become part of the network and gain access to information required to communicate within the network, both with immediate neighboring devices and with particular infrastructure devices, such as devices assuming the role of system manager or security manager of the network. As such, this entails the sub-processes described below. Note that distribution of the keying material and resource allocation steps are identical for asymmetric key-based and symmetric key-based join processes. The table of roles and their respective bitmap assignment are defined in Annex B.

The enrollment process includes:

- Network membership enrollment. A device and a security manager engage in a mutual entity authentication protocol based on asymmetric key techniques. This protocol provides evidence regarding the true device identity of both joining device and security manager, based on authentic asymmetric keys. In addition, admission may be based on non-cryptographic acceptability criteria (e.g., via a membership test of the device via an access control list). If the device has been positively authenticated and is authorized to join the network, it may be admitted to the network. The entity authentication protocol also results in the establishment of a shared key between joining device and security manager, thereby facilitating ongoing secure and authentic communications between these devices.
- Distribution of keying material. A security manager allocates keying material to a newly admitted device, so as to facilitate subsequent communications and continuous authentication of the device to other members of the network as a legitimate network device. The keying material may include DL keys, which are used to evidence network membership amongst devices in the network, and session keys, which are used to secure and authenticate ongoing communications between a newly admitted device and a system manager.

The join process assumes that devices have been endowed with sufficient information to allow proper device authentication. A joining device may have been endowed with non-security related information as well.

The asymmetric-key key agreement scheme is specified in 7.4.6.2, the key distribution scheme is specified in detail in 7.4.6.3, the resource allocation scheme is specified in detail in 6.3.9, and the asymmetric-key based join protocol is specified in 7.4.6.4. The integers, octets and entities used in the asymmetric-key based join protocol are defined in Annex F. The asymmetric-key cryptographic building blocks are defined in Annex H.

### **7.4.6.2 Asymmetric-key key agreement scheme**

#### **7.4.6.2.1 Overview**

##### **7.4.6.2.1.1 General**

Network membership enrolment is based on the execution of the asymmetric-key key agreement scheme specified in H.4.2 and involves device authentication based on implicit certificates, as specified in H.5.1. Both schemes involve asymmetric-key techniques using elliptic curves.

In the remainder of this subclause, we give an informal description of the scheme (7.4.6.2.1.2), discuss its security properties (7.4.6.2.3), and discuss formats of protocol messaging (0).

##### **7.4.6.2.1.2 Format of implicit certificate**

The implicit certificate is a proof of identity and is used in the asymmetric key join process. It can hold arbitrary data structure; however, to allow interoperability, the format of the implicit certificate, used in this standard, is defined in Table 68.

**Table 68 – Implicit certificate format**

Element name	Element identifier	Element type
Publickey_Reconstruction_Data	1	Type: OctetString (length 37 octets)
Subject	2	Type: EUI-64
Issuer	3	Type: EUI-64
Usage_Serial	4	Type: Usage_Serial structure (see Table 69)
ValidNotBefore	5	Type: TAITimeRounded
ValidNotAfter	6	Type: TAITimeRounded

- Publickey\_Reconstruction\_Data: Parameter for generating a public key with using CA's public key.
- Subject: EUI-64 of a device whose public/private key is associated with Publickey\_Reconstruction\_Data
- Issuer: EUI-64 of a device that has generated this certificate.
- Usage\_Serial: Indicating a certification usage and serial number.
- ValidNotBefore: Absolute TAI time (in second) when this certificate becomes valid.
- ValidNotAfter: Absolute TAI time (in second) when this certificate becomes invalid.

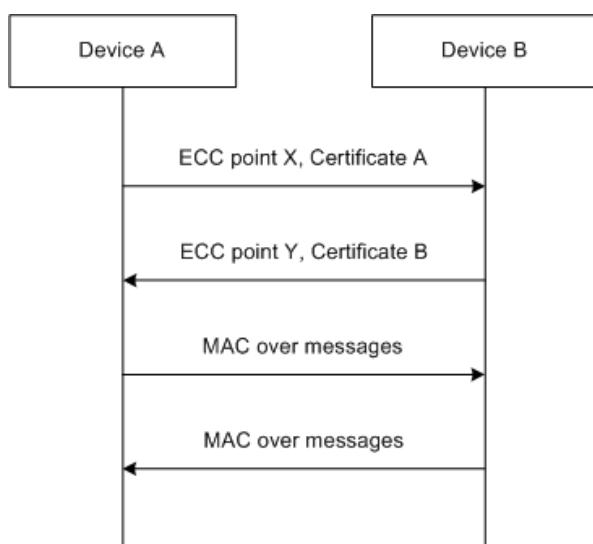
**Table 69 – Usage\_Serial structure**

Octets	Bits								
	7	6	5	4	3	2	1	0	
1	Reserved	Issuable	Serial						

- Reserved: Reserved field should be 0.
- Issuable: If this field is 0, the key pair corresponding this certificate shall not be used signed another certificate, Otherwise, the key pair may be used to sign another certificate.
- Serial: Serial number of this certificate managed by the issuer.

#### 7.4.6.2.2 Description of the scheme

Figure 46 illustrates the messaging involved in the asymmetric-key key agreement scheme used with this standard.

**Figure 46 – Asymmetric key-authenticated key agreement scheme**

In the context of the join protocol, the key agreement scheme involves messaging between a joining device and a security manager, whereby the joining device initiates the protocol and whereby the security manager acts as the so-called responder. Thus, in terms of Figure 46, the joining device assumes the role of device A and the security manager assumes the role of device B.

The protocol includes the following steps:

- 1) Key contributions. Each party randomly generates a short-term (ephemeral) public key pair and communicates the ephemeral public key (but not the private key) to the other party. In addition, each party communicates the certificate of its long-term (static) public key to the other party.
- 2) Key establishment. Each party computes the shared key based on the static and ephemeral elliptic curve points it received from the other party, and also based on the static and ephemeral private keys it generated itself. Due to the properties of elliptic curves, either party arrives at the same shared key.
- 3) Key authentication. Each party verifies the authenticity of the long-term static key of the other party, to obtain evidence that the only party that may be capable of computing the shared key is indeed the perceived communicating party.
- 4) Key confirmation. Each party computes and communicates a message authentication check value over the strings communicated by the other party, to evidence possession of the shared key to the other party. This confirms to each party the true identity of the other party and proves that the other party successfully computed the shared key. This key

confirmation message may authenticate an additional string communicated by the party itself as well. The strings and string operations are defined in Annex F.

The protocol assumes that each party has access to the root key of the certificate authority (CA) that signed the certificate received from the other party.

#### **7.4.6.2.3 Security properties of the scheme**

Successful execution of the complete scheme results in security properties, including the following:

- Mutual entity authentication. Each party has assurances as to the true identity of the other party and that that party was alive during the execution of the protocol.
- Mutual implicit key authentication. Each party has assurances that the only party that may have been capable of computing the shared key is indeed the intended communicating party.
- Mutual key confirmation. Each party has evidence that its intended communicating party successfully computed the shared key.
- Perfect forward secrecy. Compromise of the static key does not compromise past shared keys.
- No unilateral key control. Each party has assurance that neither party was able to control or predict the value of the shared key.
- Additional security properties, such as unknown key-share resilience and known-key security. For details, See Table H.2 in ANSI X9.63-2001.

The security services provided by each scheme are assured after successful completion of the complete scheme in question (and if the prerequisites of the scheme are satisfied). From the schemes themselves, it is not clear a priori what properties are provided during execution of the protocol steps of the scheme. The security services provided include:

- Processing of random key contributions does not offer any security services, since these messages are independent.
- From the perspective of the joining device A, the protocol is finished after completion of the processing steps resulting from receipt of key confirmation message  $MAC_B$ , whereas from the perspective of the security manager B, the protocol is only finished after completion of the processing steps resulting from receipt of key confirmation message  $MAC_A$ . In particular, security manager B does not have any assurances prior to receipt and processing of key confirmation message  $MAC_A$ . Thus, any actions by B triggered prior to completion of the entire protocol with A are premature, in the sense that these cannot logically be based on any security assurances (as there are none). In contrast, any actions by B triggered after successful completion of the entire protocol with A may be well-founded, in the sense that these may be based on the security services resulting from the completion of the protocol.

NOTE This re-emphasizes the importance of considering the effect of cryptographic schemes in their entirety.

#### **7.4.6.3 Key distribution scheme**

##### **7.4.6.3.1 Overview**

Key distribution is based on the shared key resulting from the asymmetric-key key agreement scheme executed between the joining device and the security manager, as described in 7.4.6.2.

##### **7.4.6.3.2 Description of the scheme**

The mechanism for distribution of keying material from the security manager to the newly joined device and the system manager is the same as that described in the symmetric-key join process. For details, see 7.4.4.

##### **7.4.6.3.3 Security properties of the scheme**

Successful execution of the key distribution scheme results in security properties including the following:

- Secure and authentic transfer of the DL key and associated keying information from the security manager to the newly joined device.
- Secure and authentic transfer of the session key and associated keying information from the security manager to the newly joined device and to the system manager selected by the security manager.
- In either case, the distributed keying material is generated by the security manager, thereby offering unilateral key control.

#### **7.4.6.3.4 Formats of protocol messaging**

The mechanism for distribution of keying material from the security manager to the newly joined device and the system manager is the same as that described in the symmetric key join process. For details, see 7.4.4.

#### **7.4.6.4 Asymmetric key-based join protocol**

The asymmetric key-based join protocol can be viewed as a protocol that combines the asymmetric-key key agreement scheme discussed in 7.4.6.2 and the key distribution scheme discussed in 7.4.6.3, the main difference being in the actual organization of messaging in frames.

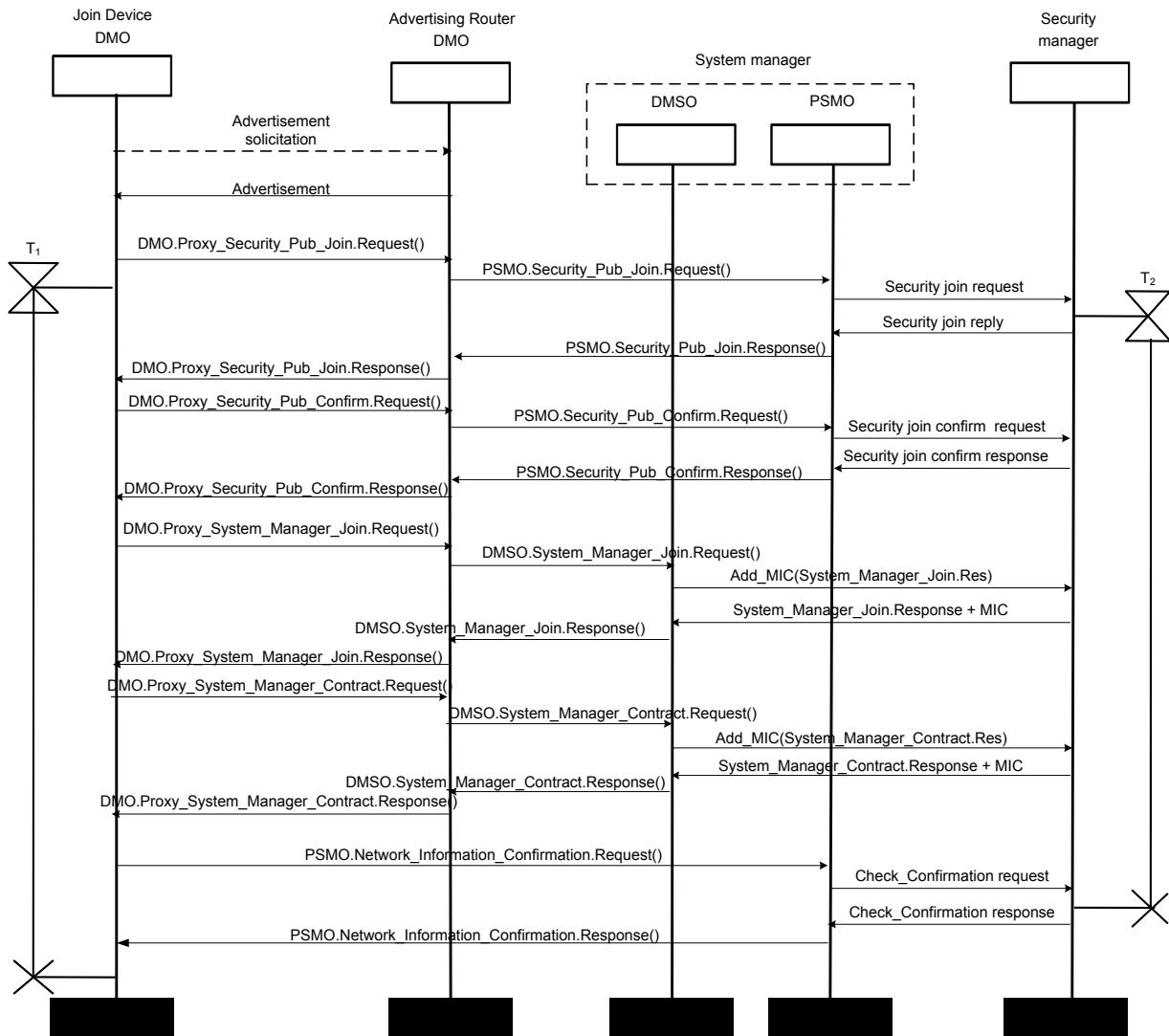
The asymmetric key-based join protocol and the symmetric key-based join protocol only differ in the use of an asymmetric-key key agreement scheme, rather than a symmetric-key key agreement scheme. Thus, all other aspects of the specification of the symmetric key-based join protocol (see 7.4.4) apply to the asymmetric key-based join protocol as well.

#### **7.4.6.5 Asymmetric key join process messages**

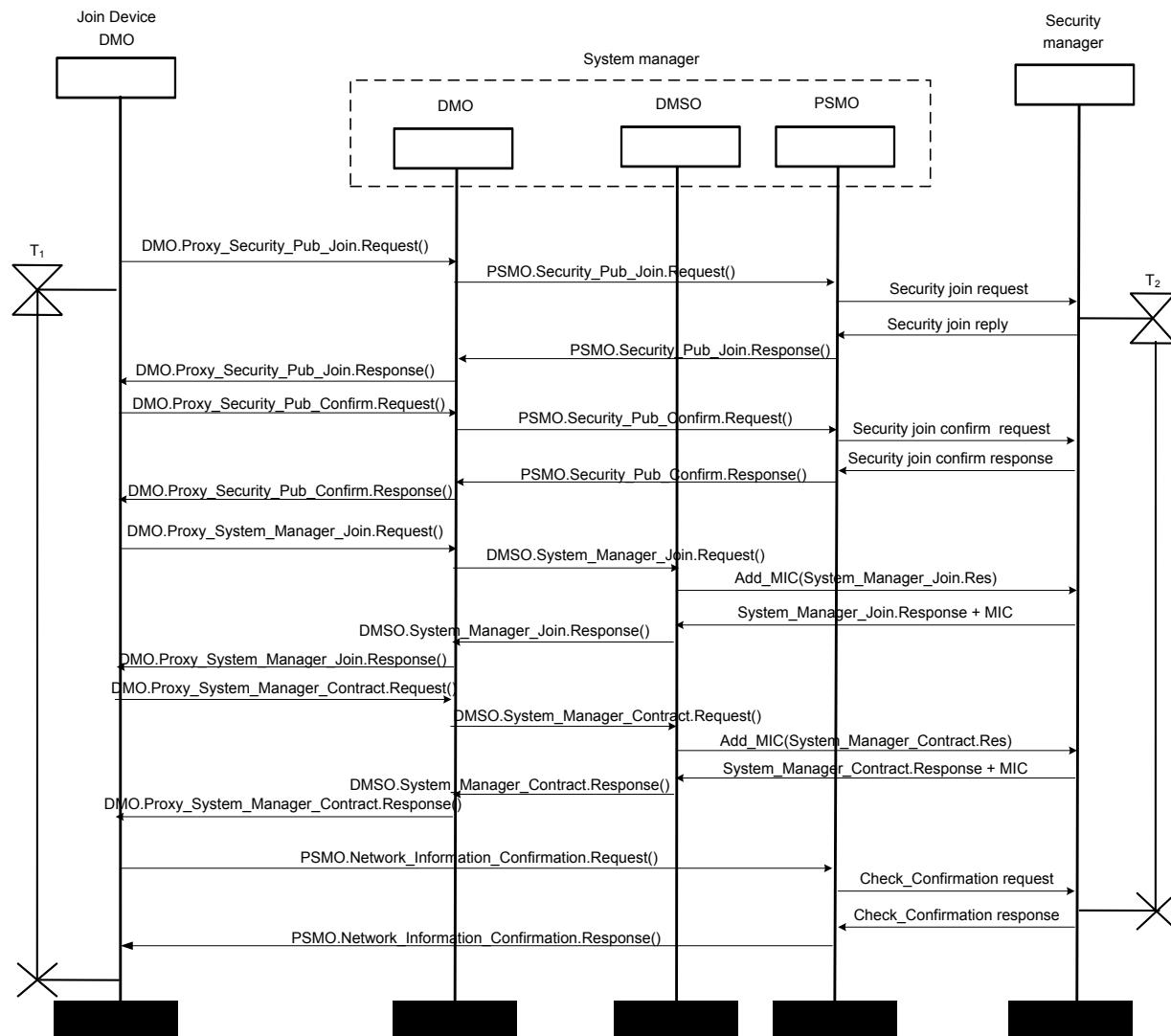
##### **7.4.6.5.1 General**

Figure 47 and Figure 28 illustrate the messaging involved in the asymmetric key join process by which a new device shall join an operating network in which it has not recently been a participant. The flow shows the normal case in which no errors or timeouts occur. The timeouts are specified in 7.4.7.3.

On the joining device, the asymmetric key join process shall be initiated by transmitting DMO.Proxy\_Security\_Pub\_Join().Request and finalized by receiving a valid PSMO.Network\_Information\_Confirmation().Response. On the security manager, the asymmetric key join process shall be initiated by receiving valid message derived from PSMO.Security\_Pub\_Join().Request and finalized by a transmitting message derived to be PSMO.Network\_Information\_Confirmation().Response.



**Figure 47 – Example: Overview of the asymmetric key join process for a device with a data link layer**



**Figure 48 – Example: Overview of the asymmetric key join process of a backbone device**

As shown in Figure 47, a new device shall use the methods defined for the advertising router's DMO to send and receive the join request and join response messages. The methods related to the non-security information are described in 6.3.9.2. The DMO methods related to the security information for the asymmetric join method are described in 7.4.6.5.2.

The advertising router shall use the methods defined for the system manager's DMSO to send and receive the non-security related join request and response messages. These DMSO methods are described in 6.3.9.5. Methods defined for the system manager's proxy security management object (PSMO) shall be used by the advertising router to send and receive the security related join request and response messages. The PSMO methods related to the security information are described in 7.4.5.2.

#### 7.4.6.5.2 Device management object and proxy security management object methods related to the asymmetric key join process

The new device shall use the `Proxy_Security_Pub_Join` method defined for the advertising router's DMO in the advertising router to send its security information that is part of the join request and to get its security information that is part of the join response. After transmitting the `DMO.Proxy_Security_Pub_Join().Request`, the new device shall start the join timer  $T_1$ . After receiving the `DMO.Proxy_Security_Pub_Join().Request`, the security manager shall start the join timer  $T_2$ .

Table 70 describes the `Proxy_Security_Pub_Join` method.

**Table 70 – Proxy\_Security\_Pub\_Join method**

<b>Standard object type name: DMO (Device management object)</b>			
<b>Standard object type identifier: 127</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Proxy_Security_Pub_Join	6	Method to use advertising router as proxy to send security join request and get security join response	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Join_Request	Data type: Security_Pub_Join_Request; see 7.4.6.5.3	Security join request based on public keys from new device that needs to be forwarded to security manager.
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Join_Response	Data type: Security_Pub_Join_Response; see 7.4.6.5.3	Security join response based on public keys from security manager that needs to be forwarded to new device; this is protected using the join key.

The advertising router shall use the Security\_Pub\_Join method defined for the system manager's PSMO for sending the security information that is part of the join request on behalf of the new device and to get the security information that is part of the join response.

The source object of the DMO.Proxy\_Security\_Pub\_Join().Request shall be the DMO in the joining device's DMAP.

Table 71 describes the Security\_Pub\_Join method.

**Table 71 – Security\_Pub\_Join method**

<b>Standard object type name: PSMO (Proxy security management object)</b>			
<b>Standard object type identifier: 105</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Security_Pub_Join	3	Method to use the PSMO in the system manager to send security join request and get a security join response	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Join_Request	Data type: Security_Pub_Join_Request; see 7.4.6.5.3	Security join request from new device to security manager.
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Join_Response	Data type: Security_Pub_Join_Response; see 7.4.6.5.3	Security join response from security manager to new device that is protected using the join key.

After receiving the Proxy\_Security\_Pub\_Join().Response message, the new device shall use the Proxy\_Security\_Pub\_Confirm() method defined for the advertising router's DMO for sending a security confirmation to the advertising router. Table 72 describes this method. The

source object of the DMO.Security\_Pub\_Join().Request shall be the DMO in the joining device's DMAP.

The advertising router shall use the Security\_Pub\_Confirm method defined for the system manager's PSMO for sending this security confirmation to the security manager. Table 73 describes this method.

The security manager is responsible to check the confirmation message. If the test fails, join state and cached information for the new device shall be initialized or dropped. If the test succeeds, then the security manager stops the join timer  $T_2$ , and sends a confirmation response and the non-security information responses to the new device.

If the new device receives a valid response against its confirmation request, then the new device stops the timer  $T_1$ .

**Table 72 – Proxy\_Security\_Pub\_Confirm method**

<b>Standard object type name: DMO (Device management object)</b>			
<b>Standard object type identifier: 127</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Proxy_Security_Pub_Confirm	7	Method to use advertising router as proxy by new device for sending security confirmation	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Security_Pub_Confirm_Request	Data type: Security_Pub_Confirm_Request; see 7.4.6.5.3	Security confirmation from new device to security manager through advertising router
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Security_Pub_Confirm_Response	Data type: Security_Pub_Confirm_Response; see 7.4.6.5.3	Security confirmation from security manager to new device through advertising router

**Table 73 – Security\_Pub\_Confirm method**

<b>Standard object type name: PSMO (Proxy security management object)</b>			
<b>Standard object type identifier: 105</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Security_Pub_Confirm	4	Method to send security confirmation of the new device to the security manager through the PSMO	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Security_Pub_Confirm_Request	Data type: Security_Pub_Confirm_Request; see 7.4.6.5.3	Security confirmation from new device to security manager through advertising router.
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Security_Pub_Confirm_Response	Data type: Security_Pub_Confirm_Response; see 7.4.6.5.3	Security confirmation from security manager to new device through advertising router.

After receiving the DMO.Proxy\_System\_Manager\_Join() and DMO.Proxy\_System\_Manager\_Contract() response, the join device invokes PSMO.Network\_Information\_Confirmation() method. Table 74 describes the method.

The Confirm field in PSMO.Network\_Information\_Confirmation().Request is the MACTag generated with following operation.

$$\text{MACTag} = \text{HMAC-MMOK\_join}[\text{MIC}_1 \parallel \text{MIC}_2 \parallel \text{Challenge}_{\text{joining\_device}}]$$

Where:

- MIC<sub>1</sub>: MIC field in DMO.Proxy\_System\_Manager\_Join().Response (see Table 19)
- MIC<sub>2</sub>: MIC field in DMO.Proxy\_System\_Manager\_Contract().Response (see Table 20)

**Table 74 – Network\_Information\_Confirmation method**

<b>Standard object type name: PSMO (Proxy Security Manager Object)</b>			
<b>Standard object type identifier: 105</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Network_Information_Confirmation	5	Method to make sure that correct network information was received by Join Device.	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Confirm	Data type: Unsigned128	Confirmation message to make sure the join device received correct network information from the system manager.
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
-	-	-	-

#### 7.4.6.5.3 Formats of protocol messaging

##### 7.4.6.5.3.1 Format of the join request internal structure (PK-join-1)

###### 7.4.6.5.3.1.1 General

The Security\_Pub\_Join\_Request data type used in the Security\_Pub\_Join method and Proxy\_Security\_Pub\_Join method has the following structure and represents the first message flow of the asymmetric-key key agreement scheme (7.4.6.2). This data type is used by the new device and its proxy router in the corresponding methods of the DMO of the proxy router and PSMO of the system manager respectively. The PK-join-1 data shall be formatted as illustrated in Table 75.

**Table 75 – Format of asymmetric join request internal structure**

<b>Standard data type name: Security_Pub_Join_Request ( PK-Join -1)</b>		
<b>Standard data type code: 415</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
New_Device_EUI64	1	Type: Unsigned64 Classification: Constant Accessibility: Read only Default value : N/A Valid value set: N/A
Protocol control field	2	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value : 10000000 Valid value set: N/A
Ephemeral elliptic curve point X	3	Type: OctetString (37 octets) Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
Implicit certificate of new device	4	Type: OctetString (length of 37-66 octets) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A

NOTE 1 The format of implicit certificate is described in 7.4.6.2.1.2.

NOTE 2 The total size of the asymmetric join request ranges from 83 to 112 octets. If the user chooses this option, the request might be fragmented into more than one NPDU.

#### **7.4.6.5.3.1.2 Format of protocol control field**

The protocol control field is 1 octet in length and specifies which algorithm is used for the asymmetric-key based join protocol and which instance of the protocol is currently being executed. This subfield shall be formatted as specified in Table 76.

**Table 76 – Format of the protocol control field**

<b>Bits: 0-5</b>	<b>6-7</b>
Reserved	Protocol sequence number

#### **7.4.6.5.3.1.3 Protocol sequence number**

The protocol sequence number is 2 bits in length and indicates the current instantiation of the protocol. Asymmetric Key Join Request  $b_1b_0 = '00'$ , Asymmetric Key Join Response  $b_1b_0 = '01'$ , Asymmetric Key Join Confirm Request  $b_1b_0 = '10'$ , Asymmetric Key Join Confirm Response  $b_1b_0 = '11'$ .

#### **7.4.6.5.3.2 Format of the asymmetric join response internal structure (PK-join-2)**

The Security\_Pub\_Join\_Response data type used in the Security\_Pub\_Join method and Proxy\_Security\_Pub\_Join method has the following structure and represents the first message flow of the asymmetric-key key agreement scheme (7.4.6.2). The PK-join-2 data shall be formatted as illustrated in Table 77.

**Table 77 – Format of asymmetric join response internal structure**

<b>Standard data type name: Security_Pub_Join_Response ( PK-Join -2)</b>		
<b>Standard data type code: 416</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
New_Device_EUI64	1	Type: Unsigned64 Classification: Constant Accessibility: Read only Default value : N/A Valid value set: N/A
Protocol control field	2	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value : 10000001 Valid value set: N/A
Ephemeral elliptic curve point Y	3	Type: OctetString (length 37 octets) Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
Implicit certificate of security manager	4	Type: OctetString (length of 37-66 octets) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A

NOTE 1 The format of the implicit certificate used in this standard is defined in 7.4.6.2.1.2.

NOTE 2 The total size of the asymmetric join response ranges from 83 to 112 octets. If the user chooses this option, the request may be fragmented into more than one NPDU.

#### **7.4.6.5.3.3 Format of the first join confirmation internal structure (PK-join-3)**

The Security\_Pub\_Confirm\_Request data type used in the Security\_Pub\_Confirm method and Proxy\_Security\_Pub\_Confirm method has the following structure and represents the third message flow of the asymmetric-key key agreement scheme (7.4.6.2). The PK-join-3 data shall be formatted as illustrated in Table 78.

**Table 78 – Format of first join confirmation internal structure**

<b>Standard data type name: Security_Pub_Confirm_Request ( PK-Join -3)</b>		
<b>Standard data type code: 417</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
New_Device_EUI64	1	Type: Unsigned64 Classification: Constant Accessibility: Read only Default value : 10000010 Valid value set: N/A
Protocol control field	2	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value : 10000010 Valid value set: N/A
Message_authentication_tag_MAC	3	Type: OctetString (length 16 octets) Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
Length of text	4	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : 0 Valid value set:0-255
Text	5	Type: OctetString up to a maximum of 31 octets (length : see element Length of Text) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A

The Message\_authentication\_tag\_MAC is generated with following formula.

$$\text{Message\_authentication\_tag\_MAC} = \text{MACmackey}(02_{16} || U || V || QEU || QEV)$$

Where:

- U is the EUI-64 of new device
- V is the 8 octet ID of the security manager
- QEU is the octet string of the ephemeral public key of the new device
- QUV is the octet string of the ephemeral public key of the security manager

This is part of the ECMQV key agreement scheme. The MACmackey is defined in ANSI X9.63 5.7. In this specification, the keyed hash function HMAC-MMO with the master key shall be used for the MACmackey function.

The text field is used to store optional information that needs to be authenticated. Users may use this field for any information to be protected during the asymmetric key join process.

#### **7.4.6.5.3.4 Format of the second join confirmation internal structure**

The Security\_Pub\_Confirm\_Response data type used in the Security\_Pub\_Confirm method and Proxy\_Security\_Pub\_Confirm method has the following data structure and represents the fourth message flow of the asymmetric-key key agreement scheme (7.4.6.2). The PK-join-4 data shall be formatted as illustrated in Table 79.

**Table 79 – Format of join confirmation response internal structure**

<b>Standard data type name: Security_Pub_Confirm_Response ( PK-Join -4)</b>		
<b>Standard data type code: 418</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
Protocol control field	1	Type: Unsigned8 Classification: Constant Accessibility: Read only Default value : 10000011 Valid value set: N/A
Message_authentication_tag_MAC	2	Type: OctetString (length 16 octets) Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
Length of Text	3	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : 0 Valid value set: 0-255
Text	4	Type: OctetString (length : see element 3) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Master_Key_HardLifeSpan	5	Type: OctetString (size: 2 Octets) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Encrypted DL key	6	Type: Unsigned 128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
DL Crypto Key Identifier	7	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
DL_Key_HardLifeSpan	8	Type: OctetString (size: 2 octets) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Encrypted system manager session key	9	Type: Unsigned 128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
System_Manager_Session_Key_HardLifeSpan	10	Type: OctetString (size: 2 octets) Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A

The Message\_authentication\_tag\_MAC is generated with the following formula.

$$\text{Message\_authentication\_tag\_MAC} = \text{MACmackey}(03_{16} || U || V || QEU || QEV)$$

Where:

- U is the EUI-64 of new device
- V is 8 octet ID of the security manager

- QEU is the octet string of the ephemeral public key of the new device
- QUV is the octet string of the ephemeral public key of the security manager

This is part of ECMQV key agreement scheme. The MACmackey is defined in ANSI X9.63 5.7. In this specification HMAC-MMO with master key shall be used for MACmackey function.

The text field is used to store optional information that needs to be authenticated. Users may use this field for any information to be protected during the asymmetric key join process.

The key material and policy fields are the same as for the symmetric key join process. Specifically, the following shall be the same as in 7.4.5:

- Master Key compressed policy
- Encrypted DL key
- DL Crypto Key Identifier
- DL key compressed policy
- Encrypted system manager session key
- System manager session key compressed policy

#### **7.4.7 Join process and device lifetime failure recovery**

##### **7.4.7.1 General**

At any point during the join process, there is a possibility that a PDU will be dropped. In this case, the system should be able to recover and proceed. The following state definition and transition outline the recovery mechanism, along with triggered side effects.

##### **7.4.7.2 Device states during the join process and device lifetime**

The device states during the join process are:

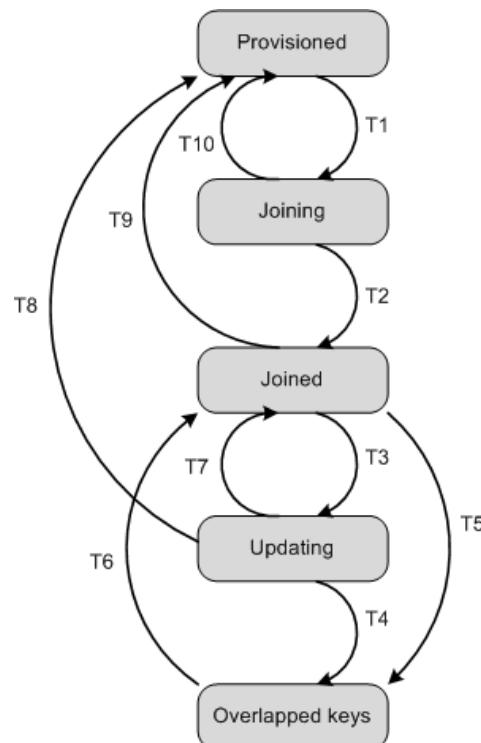
- Provisioned: No master key, not in process to get master key.
- Joining: No master key, in process to get master key.
- Joined: Having current master key, not in process to get next master key.
- Updating: Having current master key, in process to get next master key.
- Overlapped: Having current and next master key.

##### **7.4.7.3 State transitions**

The state transitions for a device joining the network shall be as outlined in Table 80 and Figure 49. The timeout values for the join process join\_timeout (ID 4) in Table 93, shall be configurable using DL advertisements. Erasing of keys should be at least equivalent to clearing of confidential data, as defined in table 2-1 of NIST SP800-88 rev 1.

**Table 80 – Join process and device lifetime state machine**

Transition	Current state	Event(s)	Action(s)	Next state
T1	Provisioned	DMO initiates the join process	Advertising Router DMO.Proxy_Security_Sym_Join().Request or DMO.Proxy_Security_Pub_Join().Request	Joining
T2	Joining	DMO.Proxy_Security_Sym_Join() .Response or DMO.Proxy_Security_Pub_Join() .Response received & crypto check ok	Populate appropriate entries in DSMO and KeyDescriptor. Call PSMO.Security_Confirm().Request (may be delayed), or Call PSMO.Network_Information_confirmation() .Request	Joined
T3	Joined	SoftExpirationTime of master key expired	Call PSMO.Security_New_Session().Request with the security manager	Updating
T4	Updating	DSMO.New_Key().Request(master_key) from security manager via the PSMO && crypto check ok	Save master key material and policy. Set Key_ID of session to the value assigned by the security manager. Return a DSMO.New_Key().Response	Overlapped keys
T5	Joined	DSMO.New_Key().Request(master_key) from security manager via the PSMO && crypto check ok	Save master key material and policy. Set Key_ID of session to the value assigned by the security manager. Return a DSMO.New_Key().Response	Overlapped keys
T6	Overlapped keys	ValidNotAfter of old master key expired.	Remove expired master key.	Joined
T7	Updating	Timeout or PSMO.Security_New_Session() .Response&& crypto check ok && SESSION_DENIED	Set the next retry time	Joined
T8	Updating	ValidNotAfter of master key expired	Remove expired master key.	Provisioned
T9	Joined	ValidNotAfter of master key expired	Remove expired master key	Provisioned
T10	Joining	Timeout	Reset state machine	Provisioned

**Figure 49 – Device state transitions for join process and device lifetime**

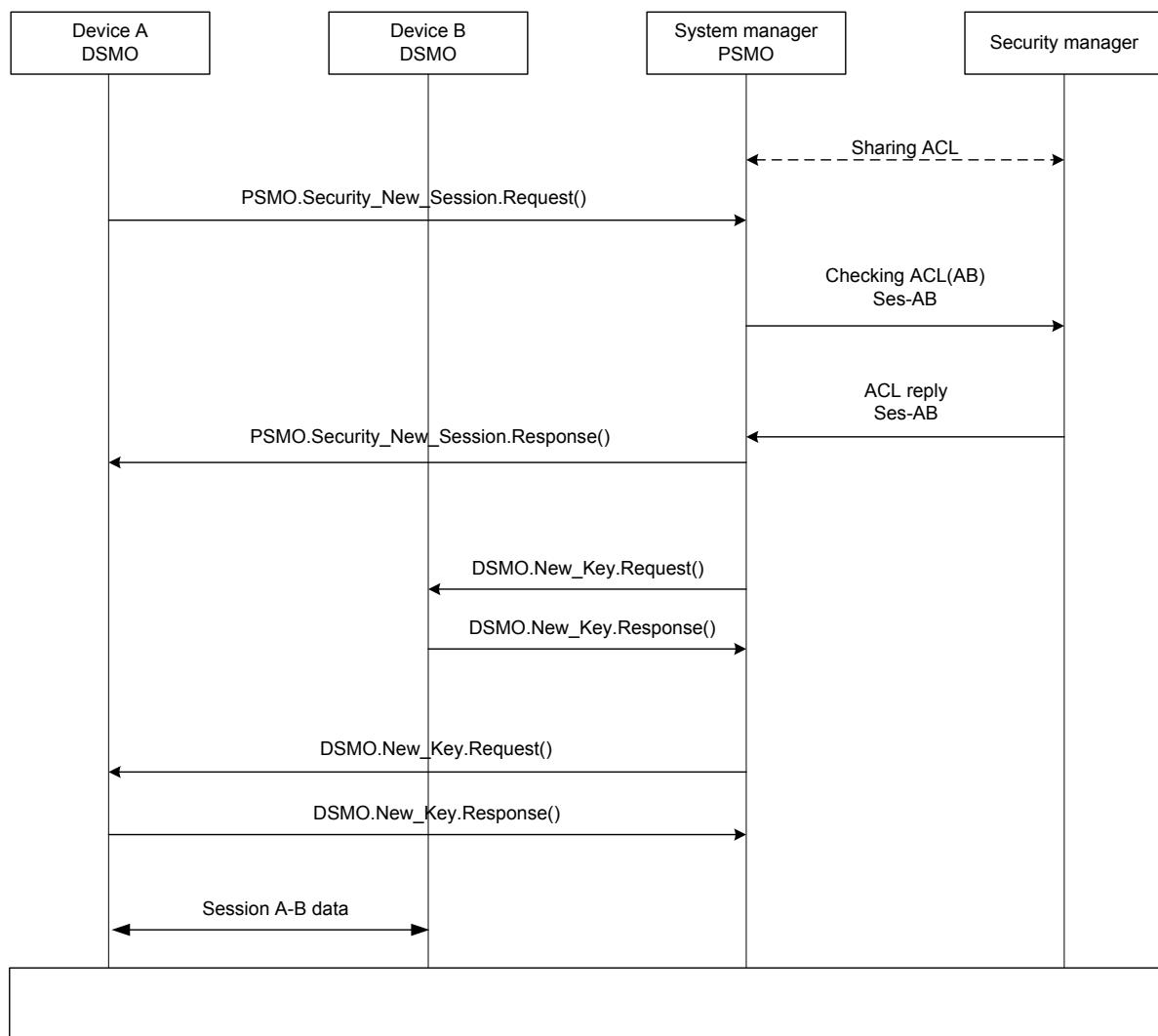
## 7.5 Session establishment

### 7.5.1 General

The session establishment occurs in support of an end-to-end secure communication between two UAPs. The end point of a session is defined as the concatenation of the network address and the transport port. The security manager is responsible for granting or denying the cryptographic material used to establish the end-to-end secure channel between the two devices.

### 7.5.2 Description

Figure 50 provides a high-level example of session establishment.



**Figure 50 – High-level example of session establishment**

In the high-level example shown in Figure 50, a UAP on device A establishes a session with a UAP on device B. The DSMO of device A sends the request to the security manager via the PSMO object of the system manager. The system manager then forwards the request to the security manager, who authenticates the request and may perform a check to verify if the session is allowed. If the session is granted, the security manager generates a single session key for both end points, encrypts a copy for device A and another copy for device B, and forwards the messages to the system manager's PSMO. The system manager's PSMO can then send the response to the session request. The system manager's PSMO then calls the DSMO method to add a new session key on device A and device B.

The session establishment may be initiated by a field device or by the security/system manager.

- The field device initiates a session establishment using PSMO.Security\_New\_Session() in

**Table 81.**

- The system/Security manager initiates a session establishment using DSMO.New\_Key() method in Table 84.

The security manager shall assign the same key and Crypto Key Identifier among devices which participate in the secure session. In the case of overlapped keys, the security header need to convey the Crypto Key Identifier used to protect the PDU at the transmitter. At the receiver, the device looks for KeyDescriptor with the Crypto Key Identifier specified in incoming PDU security header. If the Crypto Key Identifier value does not have a match, the receiving device will not be able to decrypt and/or authenticate the incoming PDU. .

### **7.5.3 Application protocol data unit protection using the Master key**

#### **7.5.3.1 General**

The request is made from the device's DSMO to the system manager's PSMO acting as a proxy to the security manager. The APDU shall be protected with using almost the same PDU security mechanism as the TL. The cryptographic key shall be the Master key, and the nonce shall be constructed in the same manner as the TL in 7.3.3.7, but the TAITimeRounded value shall be used for the 32-bit Nominal TAI time creation field. See Table 367 for coding rules applied to TAITimeRounded values.

NOTE 1 Since the join process is done at the application layer, there is no security at the Transport Layer during that process, except confirmation messages.

Since the granularity of the Time\_Stamp field is in seconds, the cryptographic operations using Master key shall be less than once per second.

NOTE 2 If the message rate using the Master key exceeds the rate of once per second, there will be a nonce collision.

#### **7.5.3.2 Replay protection for application protocol data unit protected with Master key**

Upon reception of the APDU protected with the master key, the security procedure shall check for any nonce duplicates with a valid MIC in the nonce cache with the corresponding KeyDescriptor. If a duplicate nonce is detected, the procedure shall discard the PDU before processing the ASDU, otherwise the procedure shall store the nonce into nonce cache in the KeyDescriptor.

### **7.5.4 Proxy security management object methods related to the session establishment**

Table 81 describes the Security\_New\_Session method.

**Table 81 – Security\_New\_Session method**

<b>Standard object type name: PSMO (Proxy security management object)</b>			
<b>Standard object type identifier: 105</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Security_New_Session	6	Method to use the PSMO in the system manager to send security session request and get a security session response	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	New_Session_Request	Data type: Security_New_Session_Request; see Table 82	Security new session request from a device to security manager.
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	New_Session_Response	Data type: Security_New_Session_Response; see Table 83	Security new session response from security manager to the requesting device, protected using the master key.

The Security\_New\_Session\_Request data structure that is used to form the session request is defined in Table 82.

**Table 82 – Security\_New\_Session\_Request data structure**

<b>Standard data type name: Security_New_Session_Request</b>		
<b>Standard data type code: 420</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
128_Bit_Address_A	1	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
End_Port_A	2	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
128_Bit_Address_B	3	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
End_Port_B	4	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Algorithm_Identifier	5	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : 1 = AES_CCM* Valid value set: N/A
Protocol_Version	6	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : 1 = ISA100.11a:2009 Valid value set: N/A
Security_Control	7	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Crypto_Key_Identifier	8	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Time_Stamp	9	Type: TAITimeRounded Classification: Static Accessibility: Read only Default value : N/A Valid value set: Range of TAITimeRounded.
MIC	10	Type: OctetString Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A

This data structure consists of a plaintext section only protected using the master key shared between the requester of the session and the security manager. The EUI-64 of the requester shall be used in the nonce construction to protect this structure.

- 128\_Bit\_Address\_A shall be the 128-bit network address of the first end device (typically the source) in the session
- End\_Port\_A shall be the 16 bit port of the first end UAP (typically the source) in the session

- 128\_Bit\_Address\_B shall be the 128-bit network address of the second end device (typically the destination) in the session.
- End\_Port\_B shall be the 16 bit port of the second end UAP (typically the destination) in the session.
- Algorithm\_Identifier defines the algorithm and mode of operation supported in this session. In the current release this shall be set to 0x1 = AES\_CCM\*.
- The protocol version identifies the protocol used for this security association. In this standard, this octet shall be 0x01.
- Security\_Control shall be as defined in 7.3.1.2. The security level is chosen from MIC-32, MIC-64 and MIC-128 with the Master key security level assigned in the join process. The Crypto Key Identifier Mode shall be  $b_1b_0 \cdot 01'$  corresponding to a Crypto\_Key\_Identifier Field length of 1 octet.
- Crypto\_Key\_Identifier shall be the Crypto\_Key\_Identifier of the master key used in protecting this structure.
- Time\_Stamp shall be the full 32-bit time used in the nonce construction.
- MIC shall be the integrity code generated by the AES\_CCM\* computation. The length of the MIC is assigned in Security\_Control field.

The nonce used to generate the MIC is formed as outlined in Table 57 with:

- EUI-64: EUI-64 address of the device transmitting the Security\_New\_Session Request message;
- Nominal TAI time: The Time\_Stamp field in the Security\_New\_Session Request message.

The Security\_New\_Session\_Response data structure that is used to form the new session response is defined in Table 83.

**Table 83 – Security\_New\_Session\_Response data structure**

<b>Standard data type name: Security_New_Session_Response</b>		
<b>Standard data type code: 421</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
Status	1	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Security_Control	2	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Crypto_Key_Identifier	3	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Time_Stamp	4	Type: TAITimeRounded Classification: Static Accessibility: Read only Default value : N/A Valid value set: Range of TAITimeRounded
MIC	5	Type: OctetString Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A

Fields include:

- Status shall be the status of the session where 0x1 = SECURITY\_SESSION\_GRANTED and 0x0 = SECURITY\_SESSION\_DENIED
- Security\_Control shall be as defined in 7.3.1.2. The security level is chosen from MIC-32, MIC-64 and MIC-128 with the Master key security level assigned during the join process. The Crypto Key Identifier Mode shall be  $b_1b_0'01'$  corresponding to a Crypto Key Identifier Field length of 1 octet.
- Crypto\_Key\_Identifier shall be the Crypto\_Key\_Identifier of the master key used in protecting this structure.
- Time\_Stamp shall be the full 32-bit time in TAITimeRounded form used in the nonce construction.
- MIC shall be the integrity code generated by the AES\_CCM\* computation. The length of the MIC is assigned in the Security\_Control field.

The nonce to generate the MIC is formed as outlined in Table 57 with:

- EUI-64: EUI-64 address of the device transmitting Security\_New\_Session Request message;
- Nominal TAI time: The Time\_Stamp field from the Security\_New\_Session Request message.

If the session is granted, the security manager via the PSMO of the system manager shall call the DSMO New\_Key method defined in 7.6.3 to write a new session key in the devices specified in the session request.

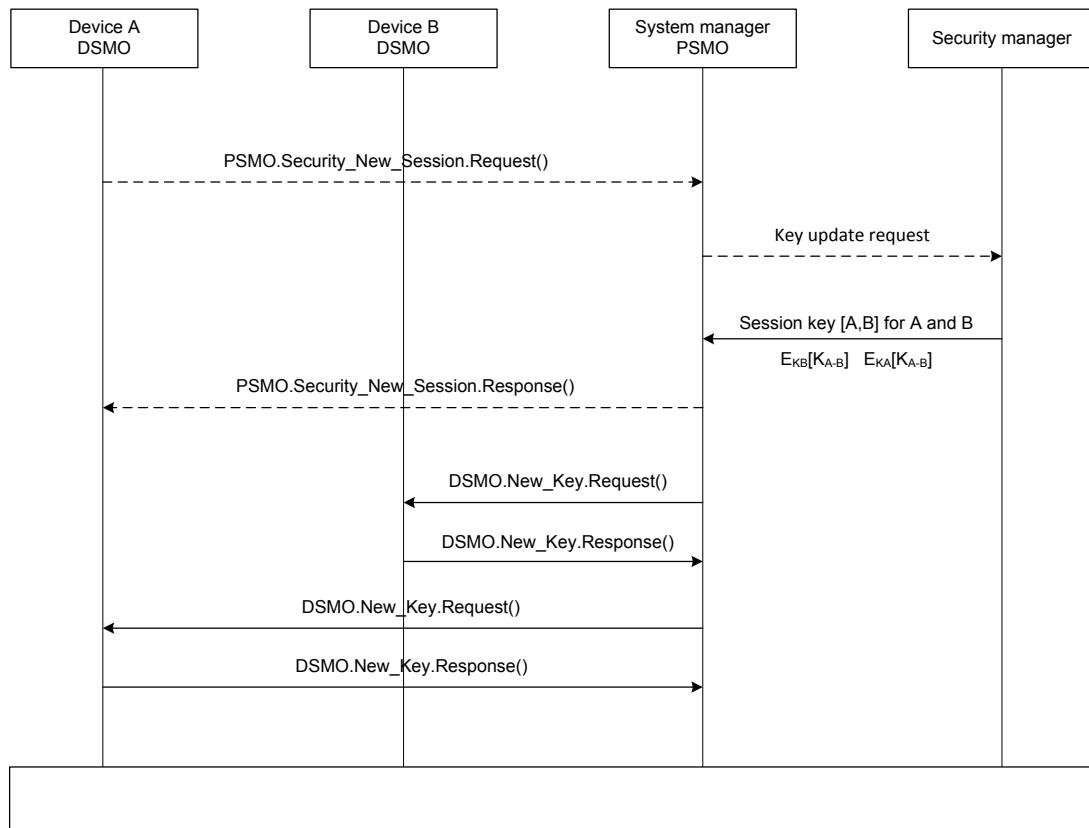
## 7.6 Key update

### 7.6.1 General

Session keys have a limited lifetime and are updated periodically to ensure that the session is kept alive. The key update process may be initiated by a device, although it should be pushed from the security manager between the SoftExpirationTime and HardExpirationTime of a session key.

### 7.6.2 Description

The key update process is summarized in Figure 51. A device participating in a session may optionally request from the security manager an update for the session key. The security manager will then issue a call to the DSMO of the end devices, via the PSMO of the system manager to update the session key on the end devices. The messages are protected under the master key shared between the security manager and each of the devices.

**Figure 51 – Key update protocol overview**

A device participating in a session may initiate the key update process by making a call to the PSMO Security\_New\_Session method. The request is forwarded from the system managers PSMO to the security manager who authenticates the request using the master key of the requesting device. If the check is successful, the security manager recognizes that the session already exists and simply proceeds with the key update protocol exactly as if the SoftExpirationTime of the session key has expired. The nonce construction for protecting APDU using master key is described in 7.5.3.

If the SoftExpirationTime of an active session key has passed, the security manager shall call the New\_Key method on the DSMO of the end devices to write a new key and accompanying policies.

The Key Update methods as described below may also be used to update the DL and Master Key.

### 7.6.3 Device security management object methods related to the session key update

Table 84 describes the New\_Key method.

**Table 84 – New\_Key method**

<b>Standard object type name: DSMO (Device security management object)</b>			
<b>Standard object type identifier: 125</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
New_Key	1	Method to use the DSMO in the device to send a protected security key and accompanying policies.	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Key_And_Policies	Data type: Security_Key_and_Policies; see Table 85	Security key and policies to be authenticated, decrypted and stored by a device

			participating in a session
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Key_Update_Status	Data type: Security_Key_Update_Status; see Table 86	Status of the key update, authenticated with the master key

The Security\_Key\_and\_Policies data structure that is used to form the New Session Key request is defined in Table 85.

**Table 85 – Security\_Key\_and\_Policies data structure**

<b>Standard data type name: Security_Key_and_Policies</b>		
<b>Standard data type code: 422</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
Key_Policy	1	Type: OctetString (see Table 89) Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
End_Port_Source (elided for DL or Master Key)	2	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
EUI64_remote (elided for DL, EUI-64 of security manager for Master Key)	3	Type: Unsigned64 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
128_Bit_Address_remote (elided for DL or Master Key)	4	Type: Unsigned128 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
End_Port_remote (elided for DL or Master Key)	5	Type: Unsigned16 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Algorithm_Identifier	6	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : 1 = AES_CCM* Valid value set: N/A
Security_Control	7	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Crypto_Key_Identifier	8	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Time_Stamp	9	Type: TAITimeRounded Classification: Static Accessibility: Read only Default value : N/A Valid value set: Range of TAITimeRounded
New_Key_ID	10	Type: Unsigned8 Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
Key_Material	11	Type: Unsigned128 Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A
MIC	12	Type: OctetString Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A

This data structure consists of a plaintext section only protected using the master key shared between the requester of the session and the security manager. The EUI-64 of the requester shall be used in the nonce construction to protect this structure.

Fields include:

- 128\_Bit\_Address\_remote shall be the 128-bit network address of the remote end device in this session. In the case of the DL Key or Master Key, this field shall be elided.
- End\_Port\_remote shall be the 16 bit port of the remote end UAP in this session. In the case of the DL Key or Master Key, this field shall be elided.
- Algorithm\_Identifier defines the algorithm and mode of operation supported in this session. In the current release this shall be set to 0x1 = AES\_CCM\*.
- Security\_Control shall be as defined in 7.3.1.2. The security level is chosen from ENC-MIC-32, ENC-MIC-64 and ENC-MIC-128 with Master key security level assigned in join process. And the Crypto Key Identifier Mode shall be b<sub>1</sub>b<sub>0</sub>'01' corresponding to a Key Index Field length of 1 octet.

NOTE It is recommended that security level for the master key is set to 128bit MIC to protect the key material strongly.

- Crypto\_Key\_Identifier shall be the Crypto\_Key\_Identifier of the master key used in protecting this structure.
- Time\_Stamp shall be the full 32-bit time in TAITimeRounded form used in the nonce construction.
- Key\_Policy shall be as described in Table 89 and populated by the security manager based on its security policies for this session.
- New\_Key\_ID shall be the 8 bit Crypto\_Key\_Identifier assigned to this key material by the security manager.
- Key\_Material shall be a 128 bit key used for this session.
- MIC shall be the integrity code generated by the AES\_CCM\* computation. The length of the MIC is specified in the Security\_Control field.

The Security\_Key\_and\_Policies data structure is protected by AES-CCM\* with following parameters:

- Authentication part: Element 1-10
- Encryption part: Element 11
- Key: Master key
- Nonce: formed Table 57 structure with;
  - EUI-64: EUI-64 address of Security manager
  - Nominal TAI Time: Time Stamp element conveyed in Security\_Key\_and\_Policies

Upon receipt of the DSMO.New\_Key().Request method call, the DSMO of the end device shall decrypt and do an integrity check on the PDU using the same incoming PDU processing step as defined in the TL (see 7.3.3.9) with the nonce constructed with the EUI-64 of the security manager and the 32 bits of time included in the PDU. The key used shall be the current master key as identified by the Crypto Key Identifier.

Upon successful completion of the check, the appropriate KeyDescriptor shall be populated using the fields in the Security\_Key\_and\_Policies data structure. In this release, the issuer is always the security manager.

The DSMO shall then generate a status message as defined in Table 86 to notify the security manager of the status of the method call.

The Security\_Key\_Update\_Status data structure that is used to form the response to the key update request is defined in Table 86.

**Table 86 – Security\_Key\_Update\_Status data structure**

<b>Standard data type name: Security_Key_Update_Status</b>		
<b>Standard data type code: 423</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
Status	1	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Security_Control	2	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Crypto_Key_Identifier	3	Type: Unsigned8 Classification: Static Accessibility: Read/write Default value : N/A Valid value set: N/A
Time_Stamp	4	Type: TAITimeRounded Classification: Static Accessibility: Read only Default value : N/A Valid value set: Range of TAITimeRounded
MIC	5	Type: OctetString Classification: Static Accessibility: Read only Default value : N/A Valid value set: N/A

Fields include:

- Status shall be the status of the session, where 0x1 = SECURITY\_KEY\_UPDATE\_FAILURE and 0x0 = SECURITY\_KEY\_UPDATE\_SUCCESS
- Security\_Control shall be as defined in 7.3.1.2. The security level is chosen from MIC-32, MIC-64 and MIC-128 with the Master key security level assigned during the join process. The Crypto Key Identifier Mode shall be b<sub>1</sub>b<sub>0</sub>'01' corresponding to a Crypto Key Identifier Field length of 1 octet.
- Crypto\_Key\_Identifier shall be the Crypto\_Key\_Identifier of the master key used in protecting this structure.
- Time\_Stamp shall be the full 32-bit time in TAITimeRounded form used in the nonce construction.
- MIC shall be the integrity code generated by the AES\_CCM\* computation. The length of the MIC is assigned in the Security\_Control field.

The nonce used to generate the MIC is formed as outlined in Table 57 with:

- EUI-64: EUI-64 address of the device transmitting the New\_Key response message;
- Nominal TAI time: The Time\_Stamp element from the Security\_Key\_Update\_Status message.

#### 7.6.4 Failure recovery

##### 7.6.4.1 General

At any point during the session establishment or key update process, there is a possibility that a PDU will be dropped. In this case, the system should be able to recover and proceed. The following state definitions and transitions outline the recovery mechanism, along with the triggered side effects.

#### 7.6.4.2 Device session and data link layer key states

Device session and DL key states include:

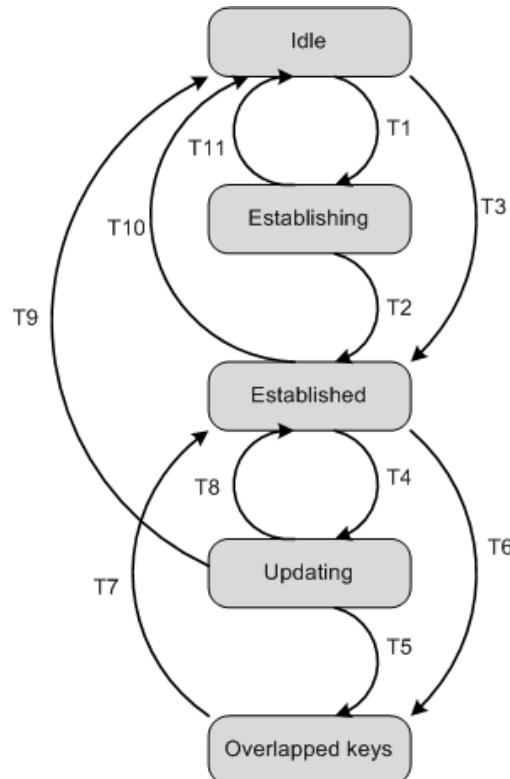
- Idle: No key, not in process to get current session key.
- Establishing: No key, in process to get current session key.
- Established: Having current key, not in process to get next key.
- Updating: Having current key, in process to get next key.
- Overlapped: Having current and next key

#### 7.6.4.3 Device session and data link layer key state transition

The state transitions shown in Table 87 and Figure 52 show the state of the device initiating a session or DL key retrieval, and of its peer accepting the session. Both start in the idle state, and both end in the established state with a valid session or DL key and relevant cryptographic elements.

**Table 87 – Session and DL key state transition**

Transition	Current state	Event(s)	Action(s)	Next state
T1	Idle	DSMO requested new session	Call the following method on the system manager: PSMO.Security_New_Session.Request()	Establishing
T2	Establishing	PSMO.Security_New_Session().Response && crypto check ok	Save key material, policy and location index, remote addr, remote port, local addr, local port as needed.	Established
T3	Idle	DSMO.New_Key().Request from security manager via the PSMO && crypto check ok	Save key material, policy and location index, remote addr, remote port, local addr, local port as needed. Return a DSMO.New_Key.Response()	Established
T4	Established	Session or DL key SoftExpirationTime expired	Call the following method on the system manager: PSMO.Security_New_Session.Request()	Updating
T5	Updating	DSMO.New_Key().Request from security manager via the PSMO && crypto check ok	Save key material, policy and location index, remote addr, remote port, local addr, local port as needed. Return a DSMO.New_Key.Response()	Overlapped keys
T6	Established	DSMO.New_Key().Request from security manager via the PSMO && crypto check ok	Store new keys in memory.	Overlapped keys
T7	Overlapped keys	ValidNotAfter of current session or DL key expired.	Remove expired key	Established
T8	Updating	Timeout OR PSMO.Security_New_Session.Response() && crypto check ok && SESSION_DENIED	Set time of next retry	Established
T9	Updating	ValidNotAfter of last session or DL key expired	Remove expired key	Idle
T10	Established	ValidNotAfter of last session or DL key expired	Remove expired key	Idle
T11	Establishing	Timeout	Reset state machine and set next retry time if necessary	Idle



**Figure 52 – Device session establishment and key update state transition**

NOTE 1 If device receives the DSMO.New\_Key() request while it is in the Updating or Overlapped state, the device will discard the master key which is not used to encrypt for new master key.

NOTE 2 If device receives the new master key which was encrypted using an unknown master key through a DSMO.New\_Key() request, the device can query for the master key for decryption with PSMO.New\_Session\_Request() request, to the security manager to re-synchronize the master keys. The security manager can use the master key which can be inferred for the encryption of the master key.

## 7.7 Security manager functionality

### 7.7.1 Proxy security management object

The attributes of the PSMO are given in Table 88.

**Table 88 – Attributes of PSMO in the system manager**

<b>Standard object type name: Proxy security management object (PSMO)</b>				
<b>Standard object type identifier: 105</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Reserved for future standard releases	1-63	-	-	-

### 7.7.2 Authorization of network devices and generation or derivation of initial master keys

The security manager maintains a database containing:

- A list of devices whose credentials have been established through a provisioning process or upon first attempt to join the network, and that have not been revoked; and
- A list of valid join keys and their associated lifetimes that have been issued to provisioning agent devices, which might be provided by new devices that attempt to join the network.

When a new device attempts to join the network and its request comes to the security manager via the system manager's PSMO as described in 7.4.5, the security manager

examines the first two lists for a quick accept/reject decision. Otherwise, the procedure shall be as described in 7.4.6.

### **7.7.3 Interaction with device security management objects**

A security manager interacts with a device's DSMO via the PSMO:

- During the join process;
- When distributing new keys;
- When receiving new keys that have been established by other devices through use of key agreement protocols;
- During a join process as described in 7.4.4; and
- During key recovery when a new network security manager replaces a failed one.

### **7.7.4 Management of operational keys**

#### **7.7.4.1 General**

The security manager maintains the current master key and associated key-generation and policy attributes for each device it manages.

All shared-secret symmetric keys are maintained in the security manager's operational storage, which in higher-security implementations should be physically protected within the security manager crypto module.

Where the participating devices do not both implement the set of asymmetric cryptography primitives specified in 7.4.6, which is a construction option for each device, the security manager may also generate shared-secret symmetric data keys for their unicast DL and TL associations.

**NOTE** Many devices do not have a high-entropy source of random bits. Without such a source, any key component generated by the device may be susceptible to inference.

#### **7.7.4.2 Key archiving**

Regulation or policy may require that keys be archived to permit concurrent or subsequent decryption of encrypted messaging.

#### **7.7.4.3 Key recovery**

The security manager should support two forms of key recovery:

- Recovery by a field device that has lost keys that were maintained in volatile storage (e.g., RAM) due to power failure or uncorrected memory error; and
- Recovery by a new network security manager of the operational keys currently in use in the network.

Each field device that supports asymmetric-key cryptography shall keep in non-volatile storage:

- Its EUI-64 address; and
- Its public/private key pair, with the latter as a signed certificate if that is available.

Each field device that supports only symmetric-key cryptography shall keep in non-volatile storage:

- Its EUI-64 address; and
- Its join key and optionally rewritten join key and related keying information if it has previously been a member of the network.

All other operational keying information may be kept in volatile storage, subject to loss upon device power failure or memory corruption, since this information can be regenerated by a security manager once that security manager has determined the EUI-64 address of the device.

#### **7.7.4.4 Security policy administration**

Primary network-wide security policy deployment options are selected during initial setup of the security manager for a network. Other policy deployment options may be selected at a later time.

### **7.8 Security policies**

#### **7.8.1 Definition of security policy**

In this specification, the security policy is defined as a combination of the following parameters:

- Key Type defined in Table 90;
- Key Usage defined in Table 91;
- Key Lifetime defined in 7.2.2.4; and
- Security Level defined in 7.3.1.1.

Keys are distributed with the above parameters specified explicitly or reconstructed implicitly at the recipient. A corresponding KeyDescriptor is generated with those parameters.

#### **7.8.2 Policy extent**

Security policies constrain the security choices that individual programs and devices can make. These policies exist at the following levels:

- Subnet-wide, across all devices participating in a given subnet, which may encompass the entire networked system;
- Device-wide, across all application programs and supporting communications layers within the device;
- Key-wide, across all PDUs secured with a given key; and
- Link-wide, across all PDUs transmitted over a given connection defined by a source and a destination, which may include UAP ports, thus providing UAP-wide policies, across all service invocations by a given application.

Some system-wide policies shall be established before system operation begins; others can be changed dynamically while the system is operating, without interrupting ongoing sessions.

#### **7.8.3 Unconstrained security policy choices**

System security policy choices may be made during system operation. The new policy will go into effect with the next rekeying of the affected devices by a security manager. Thus, operation with a given symmetric key always has a fixed set of attributes.

#### **7.8.4 Policy structures**

The format of the policies is outlined in Table 89.

**Table 89 – Structure of Policy field**

<b>Octets</b>	<b>Bits</b>											
	7	6	5	4	3	2	1	0				
0	Key type			Key usage				Granularity				
1-4	Nominal ValidNotBefore											
5-6, (7,8 opt)	HardLifeSpan											
9	Security_Level			Reserved								

Fields include:

- Key\_Type: Type of key defined in Table 90.
- Key\_Usage: Usage of key defined in Table 91.
- Granularity: Unit in which Nominal HardLifeSpan is interpreted defined in Table 92.

- Nominal ValidNotBefore in seconds: Absolute TAI time in TAITimeRounded form, when the key recipient can start to use the key.
- HardLifeSpan: Duration of time for which that key is valid. The key valid duration starts from ValidNotBefore. If the ValidNotAfter field is filled with 0x00 for any granularity, that key has an infinite lifetime and thus the key will never expire. Unless ValidNotAfter is infinite, the actual time duration of the KeyHardLifeSpan must not exceed 48.5 days (see 7.3.2.4.10) in any granularity.
- Security\_Level: Security level for each key, defined in Table 35.
- Reserved: Reserved field should be 0 ( $b_4b_3b_2b_1b_0$  '00000').

The possible values for the key types are outlined in Table 90.

**Table 90 – Key types**

Key Type Value	Key Type field $b_2b_1b_0$	Description
0x00	000	Reserved
0x01	001	Symmetric-key keying material, encrypted
0x02	010	ECC manual certificate
0x03	011	ECC implicit certificate
0x04 - 0x07	100 - 111	Reserved

The possible values for the Key Usage are outlined in Table 91.

**Table 91 – Key usage**

KeyUsage value	KeyUsage field $b_2b_1b_0$	Description
0x00	000	Group key for PDU processing (i.e., DL key)
0x01	001	Link key for PDU processing (i.e., TL session key)
0x02	010	Master key for session establishment
0x03	011	Join key
0x04	100	Public-key for ECMQV scheme
0x05	101	Root key CA for ECQV scheme
0x06	110	Reserved
0x07	111	Fixed global non-secret key

The granularity of the HardLifeSpan in the key policy is outlined in Table 92.

**Table 92 – Granularity**

Granularity	Granularity field $b_2b_1b_0$	Time unit	HardLifeSpan (octets)
0x00	000	Second	4
0x01	001	Minute	3
0x02	010	Hour	2
0x03	011	Day	2

The following policies shall be available to a network specified by this standard. The variable k indicates a variable that may be set by the security manager of a given network.

- Alerts and logging
- Events triggering alerts
- Device policy
  - The DL authentication shall always be on with an authentication tag length of 32-bits. The default key used by a joining device is the well known K\_global used to detect random errors only. The higher level APDU is protected by a secret key during a secure join. See 7.4.

NOTE 1 The DL MIC length is specified in 7.3.1.1 and the constraints for the DMIC are specified in 7.3.2.

- Alerts and logging

- A device keeps track of the number of failed cryptographic computation over a period of time. If a configurable threshold is exceeded, an alert is generated. Alerts include DPDU fail rate exceeded, TPDU fail rate exceeded, and key update fail rate exceeded. See alerts in 7.11.4.

The link, security association and PDU policy are applied through the Key policy. All users of a given key shall have the same policy; see Table 89. The configurable elements include:

- Types of key; see Table 90
- Granularity of TAI time used in the key lifetime; see Table 92.
- MIC size (32, 64, 128 bits); see Table 35.
  - DMIC size of 32, 64 or 128 bits (set to 32 by default) set in the DMXHR; see 7.3.2.2.
  - TMIC size of 0, 32, 64, or 128 (set to 0 if security is off, set to 32 if security is on) Soft lifetime See Table 94.
- Payload encryption on/off.
  - DL encryption on/off (set to off by default) set in the DMXHR; see 7.3.2.2.
  - TL encryption on/off (set to off by if no security is off, set to on if security is on).
- HardLifeSpan; see Table 89.
  - Expressed as an absolute value of TAI time at the maximum lifetime from the time of key generation to prevent a rollover of the nonce.

NOTE 2 This issue is linked to the TAI time granularity in the nonce (see 6.3.10; 48.5 days if used at 1 024 PDUs/s).
- Key originator. The EUI-64 of the generator of a given key (typically the security manager) which shall set the policy for a given key.
- Allowed sessions in the security manager

The access control function for a session establishment is required only in the Security manager. The security manager decides to grant or deny a session in the session establishment phase. The result is returned by the PSMO.Security\_New\_Session() method. If a security manager has both an Allowed and a Disallowed list, the security manager may indicate which one has precedence.

- Allowed list
  - The security manager may have a list of allowed devices identified by valid information (e.g., EUI-64 and TAG name) listed in Table 425.
- Disallowed list
  - The security manager may have a list of disallowed devices identified by a valid identifier (e.g., EUI-64 and TAG name).

## 7.9 Security functions available to the application layer

### 7.9.1 Parameters on transport service requests that relate to security

UAPs are permitted to establish application associations dynamically by requesting a session to be established. See 6.3.11.2.5.2. After a session is established, all communications from that UAP with those peers is handled securely until a period of non-use or until a need to reuse storage for security state information causes the transport layer to terminate the prior transport security association. If a subsequent transport service request from the UAP to those peers occurs after the transport security association has been discontinued, that subsequent request shall be treated as a new request, resulting in a new transport security association.

NOTE 1 There is intentionally no ability to carry unsecured TPDUs on the transport security association once it has been established, since such a mechanism would be trivially easy to attack simply by altering selected authenticated TPDUs to indicate that they employed no authentication.

NOTE 2 To support stateless application layer services, the least-recently-used policy may be applied by underlying layers for recycling any resource commitments (e.g., security connection state) that they might make.

NOTE 3 It would assist efficient security system operation if each transport service request on an association had the ability to hint at the expected interval before next use of the association. Such hinting provides guidance to the management of the implicit transport security connections needed for secured transport communications, permitting intelligent caching of established security connections and minimizing the thrashing that occurs when an implicit security connection is closed and then re-opened after a new key is established at all association participants.

The permitted security levels (see 7.3.1.1) on a transport service request are:

- Encryption of the TPDU upper-layer payload, on/off.
- Authentication of the TPDU with a TMIC of size 32, 64 or 128 bits.

NOTE 4 In an API, these may be conveyed jointly as a single signed integer (e.g., an Integer8), where the sign was used to designate encryption (–) or not (+), and the magnitude was used to specify the requested size nn, with the value zero representing a request for no authentication and no encryption.

## **7.9.2 Direct access to cryptographic primitives**

### **7.9.2.1 General**

UAPs may use any of the cryptographic services available to a device. These include:

- Unkeyed and keyed hash functions.
  - Pseudo-random or true-random bit string generation.
  - Symmetric key cryptography.
  - Block cipher encryption.
- NOTE 1 Exclusion of block cipher decryption makes it more likely that implementation can be hardware-assisted.
- Stream cipher functions for processing data strings that include authentication, encryption, and extended authentication with encryption, decryption, and decryption with extended authentication.

The available cryptographic primitives also may include a single construction option:

- Asymmetric key cryptography.
- Encryption with a public key and decryption with a private key of a private/public key pair.
- Signing with a private key and signature authentication with a public key of a private/public key pair.
- Key pair generation.
- Certificate generation, signing, and self-signing.
- Two-party Menezes-Qu-Vanstone key agreement.
- Pintsov-Vanstone signatures.

NOTE 2 The single construction option asymmetric-key cryptography provides all of these capabilities.

Abstract service definitions for all of the primitives of 7.9.2 are specified in 6.2.3.

### **7.9.2.2 Unkeyed hash functions**

A secure unkeyed (or fixed-key) one-way hash function shall be provided.

The default unkeyed hash shall be Matyas-Meyer-Oseas (MMO) as specified in ISO/IEC 10118-2, based on the block cipher of 7.9.3.2.

NOTE 1 Use of the MMO algorithm makes it more likely that implementation can be hardware-assisted.

Other unkeyed hash functions may be used where needed, either due to national requirement or because a larger output hash size is required for some application or to counter a threat.

NOTE 2 An alternate cryptographic algorithm package may be required for US government systems, because MMO is not authorized for US government use.

### 7.9.2.3 Random numbers

Each device shall provide a high-quality source of pseudo-random or random bits. The source of pseudo-random bits may be a properly-seeded generator that is compliant with ANSI X9.82 or FIPS 186-3. Where available, the high-entropy source should be a true-random bit generator.

A high quality source of random bits shall be used in the asymmetric key join. A properly seeded pseudo-random number generator may be used in generating challenge values in the symmetric key join.

NOTE 1 True-random bit generators are not suitable for direct use due to the inability to prove any statistical properties of such a source other than its non-determinism. Instead, they are used to seed and reseed pseudo-random bit generators, whose statistical properties are quantifiable. Certification of the entropy source (as the certification of the security implementation), being a highly specialized function, should be deferred to an accredited entity. The NIST document SP800-22 is useful in testing the random and pseudorandom number generators.

NOTE 2 In the symmetric key join process, it may be possible to generate a seed with using AES to encrypt the TAI time using the join key (i.e., Seed = AES[K\_join, TAI]). The join key is assuming to be high entropy, generated in the security manager and distributed during the provisioning phase.

## 7.9.3 Symmetric key cryptography

### 7.9.3.1 Keyed hash functions

The default keyed hash shall be HMAC, based on the unkeyed hash of 7.9.2.1. (See FIPS 198.)

### 7.9.3.2 Block cipher encryption and decryption functions

The default block cipher shall be AES-128, which has a 16 B block size and a 16 B key size. (See FIPS 197.)

Alternate block ciphers may be used with appropriate algorithm identifier where needed, either due to national requirement or because a larger key size or block size is required for some application or to counter some threat.

### 7.9.3.3 Stream cipher functions for encryption, decryption, authentication, extended authentication with encryption, and decryption with extended authentication

The security of this system is based in part on the availability of a stream cipher mode of operation of a block cipher that provides encryption/decryption, authentication, or both. When both are provided, the authentication can extend to data that is not included in the encryption/decryption process.

NOTE Encryption/decryption without authentication is avoided within TPDUs and DPDUs because there are a number of published cryptanalytic attacks that apply to all such schemes. However, the encryption-only and decryption-only modes of CCM\* are available to UAPs for their use, such as for protection of data in place.

The default stream cipher mode of operation of the block cipher of 7.9.3.2 shall be CCM\* (see NIST SP800-38C). CCM\* may be used for authentication-only, for extended-authentication-with-encryption, or for decryption-with-extended-authentication.

### 7.9.3.4 Secret key generation primitive

A secret key generation (SKG) primitive shall be used by the symmetric-key key agreement schemes specified in this standard.

This primitive derives a shared secret value from a challenge owned by an entity U1 and a challenge owned by an entity U2 when all the challenges share the same challenge domain parameters. If the two entities both correctly execute this primitive with corresponding challenges as inputs, the same shared secret value will be produced.

The shared secret value shall be calculated as follows:

- Prerequisites: The prerequisites for the use of the SKG primitive are:

- Each entity shall be bound to a unique identifier (e.g., the EUI-64 of the device). All identifiers shall be bit strings of the same length. Entity U1's identifier will be denoted by the bit string U1. Entity U2's identifier will be denoted by the bit string U2.
- A specialized MAC scheme shall have been chosen, with tagging transformation as specified in 5.7.1 of ANSI X9.63-2001 [2]. The length in bits of the keys used by the specialized MAC scheme is denoted by mackyelen.
- Input: The SKG primitive takes as input:
- A bit string MACKey of length mackyelen bits to be used as the key of the established specialized MAC scheme.
- A bit string QEU1 owned by U1.
- A bit string QEU2 owned by U2.
- Actions: The following actions are taken:
- Form the bit string consisting of U1's identifier, U2's identifier, the bit string QEU1 corresponding to U1's challenge, and the bit string QEU2 corresponding to QEU2's challenge:
- MacData = U1 || U2 || QEU1 || QEU2.
- Calculate the tag MacTag for MacData under the key MacKey using the tagging transformation of the established specialized MAC scheme:
- MacTag = MACMacKey(MacData).
- If the tagging transformation outputs invalid, output invalid and stop.
- Set Z=MacTag.
- Output: The bit string Z as the shared secret value.

## **7.10 Security statistics collection, threat detection, and reporting**

Major security-related events logged by the security manager should include:

- Authorizations of new devices;
- First joining of new devices to the network; and
- Prolonged disappearance of devices from the network, particularly when they are expected to have a stationary presence.

NOTE Other logged security events may be considered in a future release of this standard.

The following security-related events shall both be logged and alerted:

- MIC failure rates on received DPDUs that appear to be properly-formed DPDUs specifying the proper network-ID, that exceed a range specified in attribute 5 of the DSMO;
- MIC failure rates on received TPDUs that exceed a range specified in attribute 6 of the DSMO ; and
- Any integrity failure detected when unwrapping a wrapped symmetric key that exceeds a range specified in attribute 9 of the DSMO.

## **7.11 Device security management object functionality**

### **7.11.1 General**

The device security management object (DSMO) is part of the DMAP and is the local security management application in each device. It is responsible for the agreement and exchange of cryptographic material along with associated policies. It communicates with the DSMO of the security manager via the proxy security manager object (PSMO) of the system manager. Therefore, transport layer security shall be used to protect the DSMO traffic, except for the join process wherein special steps need to be performed.

### **7.11.2 Device security management object attributes**

Table 93 describes the DSMO.

**Table 93 – Device security management object attributes**

<b>Standard object type name: DSMO (Device security management object)</b>				
<b>Standard object type identifier: 125</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
DPDU_MIC_Failure_Limit	1	The threshold of DPDU MIC failures per time unit beyond which an alert will be sent to the security manager. It is reset to 0 after an alert is generated	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 5 Valid value set: 0 – 65535	Set to 5 by default
DPDU_MIC_Failure_Time_Unit	2	The time interval in seconds used to determine the DPDU MIC failure rate	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 60 s Valid value set: 0 – 65535.	Set to 60 s by default
TPDU_MIC_Failure_Limit	3	The threshold of TPDU MIC failures per time unit beyond which an alert will be sent to the security manager. It is reset to 0 after an alert is generated	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 5 Valid value set: 1 – 65535	Set to 5 by default
TPDU_MIC_Failure_Time_Unit	4	The time interval in seconds used to determine the TPDU MIC failure rate	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 5 Valid value set: 0 – 65535.	Set to 5 by default
DSMO_KEY_Failure_Limit	5	The threshold beyond which an alert will be sent to the security manager. It is reset to 0 after an alert is generated	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 1 Valid value set: 0 – 65535	Set to 1 by default
DSMO_KEY_Failure_Time_Unit	6	The time interval in hours used to determine the DSMO Key failure rate	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 1 Valid value set: 0 – 65535.	Set to 1 hour by default
Security_DPDU_Fail_Rate_Exceeded_AlertDescriptor	7	Used to change the priority of Security_DPDU_Fail_Rate_Exceeded Alert that belongs to the security category. This alert can also be turned on or off	Type: Alert Report Descriptor Classification: Static Accessibility: Read/write Initial default value: Alert Report disabled = False, Alert Report Priority = 6 Valid value set: See type definition	See alert definition
Security_TPDU_Fail_Rate_Exceeded_AlertDescriptor	8	Used to change the priority of Security_TPDU_Fail_Rate_Exceeded Alert that belongs to the security category. This alert can also be turned on or off	Type: Alert Report Descriptor Classification: Static Accessibility: Read/write Initial default value: Alert Report disabled = False, Alert Report Priority = 6 Valid value set: See type definition	See alert definition
Security_Key_Update_Fail_Rate_Exceeded_AlertDescriptor	9	Used to change the priority of Security_Key_Update_Fail_Rate_Exceeded Alert that belongs to the security category. This alert can also be turned on or off	Type: Alert Report Descriptor Classification: Static Accessibility: Read/write Initial default value: Alert Report disabled = False, Alert Report Priority = 6 Valid value set: See type definition	See alert definition

		on or off	definition	
pduMaxAge	10	The maximum amount of time in seconds a PDU is allowed to stay in the network. If a PDU is received in a time window exceeding this period, it shall be rejected at the receiver	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 62 Valid value set: 0-600	Set to 510 s by default.

### 7.11.3 KeyDescriptor

#### 7.11.3.1 General

The information associated with a key is summarized in Table 94.

**Table 94 – KeyDescriptor (INFORMATIVE)**

Element name	Element identifier	Element scalar type
KeyLookupData	1	Type: OctetString Size = 36 octets Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: All 0 to all 1 See Table 95
KeyUsage	2	Type: Unsigned8 Size = 1 octet Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: 0x00 to 0x07 See Table 91
ValidNotBefore	3	Type: TAITimeRounded Uint40 Size = 5 octets Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A
SoftExpirationTime	4	Type: TAITimeRounded Size = 4 octets Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A
ValidNotAfter	5	Type: TAITimeRounded Size = 4 octets Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A
Issuer	6	Type: EUI-64 or 128-bit address Size = 16 or 8 octets Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A
CryptoKeyIdentifier	7	Type: Unsigned8 or Unsigned64 Size = 1 or 8 octets Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A
KeyMaterial	8	Type: OctetString Size = Variable Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: [0– 2 <sup>128</sup> -1]
Security level	9	Type: Unsigned8 Size = 1 octets Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: 0x00 to 0x07 See Table 35

Element name	Element identifier	Element scalar type
Counter	10	Type: Unsigned8 Size = 1 octet Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: [0– 255]
NonceCache	11	Type: OctetString Size = Variable Classification: Dynamic Accessibility: Read/write Initial default value: N/A Valid value set: N/A
MICFailures	13	Type: Unsigned16 Size = 2 octet Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: [0– 2 <sup>16</sup> -1]
* indicates an index field		

The TL KeyLookupData OctetString fields are listed in Table 95.

**Table 95 – TL KeyLookupData OctetString fields**

Field name	Field scalar type
SourceAddress	Type: 128-bit address
SourcePort	Type: Unsigned16
DestinationAddress	Type: 128-bit address
DestinationPort	Type: Unsigned16

Fields include:

- KeyLookupData
- At the TL, used as index to find a key for a given association
- At the DL, this field is not used and shall be set to all 0x00
- KeyUsage: Identifies the key as a DL key or a TL key. Using a 2 bit bitmap allows a key to be defined to be used as a DL and a TL key at the same time, if allowed by the key policy.
- ValidNotBefore: Time (TAI) at which the key becomes valid
- ValidNotAfter: Time (TAI) at which the key becomes invalid
- SoftExpirationTime: Time (TAI) at which an updated key is needed
- Issuer: Address of the issuer of the key; this can be a 128-bit address or an EUI-64
- CryptoKeyIdentifier: Crypto Key Identifier, set by the key issuer. This value is used to identify the key used when overlapping keys are valid.
- KeyMaterial: Key data for encryption/decryption and/or MIC generation
- Security Level: shall be as described in Table 89
- Counter: If Type bit0 is set to 0 (this key is not a DL key), this field is not used and shall be set to all 0
- NonceCache: If Type bit1 is set to 0 (this key is not a TL session key), this field is not used and shall be set to NULL.
- MICFailures: Number of MIC failures after which an alarm will be generated

NOTE The internal representation of the Key Descriptor is up to the implementer and is shown here as a recommendation.

### 7.11.3.2 Additional device security management object methods to support key management

Table 96 describes the delete key method. The result of the method invocation is stored into ServiceFeedbackCode in the application sub-layer header and returned to the requesting device. The nonce construction for protecting APDU using master key is described in 7.5.3.

**Table 96 – Delete key method**

<b>Standard object type name(s): Device security management object</b>				
<b>Standard object type identifier: 125</b>				
<b>Defining organization: ISA</b>				
<b>Method name</b>	<b>Method ID</b>	<b>Method description :</b>		
Delete_key	2	This method is used to delete a 128 bit symmetric AES key on a device. This method is evoked by the PSMO of the security manager. The method shall be protected by the current master key shared between the device and the security manager.		
<b>Input arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
1	KeyUsage	Unsigned8	KeyUsage defined in Table 91.	
2	Crypto_Key_Identifier	Unsigned8	The Crypto_Key_Identifier used to uniquely identify keys overlapping in validity period.	
3	Source_Port	Unsigned16	Source port; If KeyUsage is not 0x01 (i.e. TL session key), this field should be elided.	
4	Destination_Address	Unsigned128	Destination Address; If KeyUsage is not 0x01 (i.e. TL session key), this field should be elided.	
5	Destination_Port	Unsigned16	Destination Port; If KeyUsage is not 0x01 (i.e. TL session key), this field should be elided.	
6	MasterKeyID	Unsigned8	Crypto Key Identifier for the Master key used for generating MIC.	
7	Time_Stamp	Unsigned32	Time of creating this message in TAITimeRounded form. This argument is time portion of the nonce used for generating MIC to protect this method call	
8	MIC	OctetString	The integrity check using AES_CCM*. The MIC length is chosen from MIC-32, MIC-64 and MIC-128 with Master key security level assigned in join process.	
<b>Output arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	

The MIC is generated by an AES-CCM\* operation with following parameters:

- Authentication part: Element 1-7

- Encryption part: none
- Key: Master key, which has Crypto Key Identifier = MasterKeyID
- Nonce: formed Table 57 structure with;
  - EUI-64: EUI-64 address of Security manager
  - Nominal TAI Time: Time Stamp field conveyed in Delete\_Key() request.

The Key\_Policy\_Update method is described in Table 97. The result of the method invocation is stored into ServiceFeedbackCode in the application sub-layer header and returned to the requesting device. The nonce construction for protecting APDU using master key is described in 7.5.3.

**Table 97 – Key\_Policy\_Update method**

<b>Standard object type name(s): Device security management object</b>			
<b>Standard object type identifier: 125</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
Key_Policy_Update	3	This method is used to update a policy attached to a 128 bit symmetric AES key on a device. This method is evoked by the PSMO of the security manager. The method shall be protected by the current master key shared between the device and the security manager.	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	KeyUsage	Unsigned8	KeyUsage defined in Table 91
2	Crypto_Key_Identifier	Unsigned8	The Crypto_Key_Identifier used to uniquely identify keys overlapping in validity period.
3	Source_Port	Unsigned16	Source port; If KeyUsage is not 0x01 (e.g. TL session key), this field should be elided.
4	Destination_Address	Unsigned128	Destination Address; If KeyUsage is not 0x01 (e.g. TL session key), this field should be elided.
5	Destination_Port	Unsigned16	Destination Port; If KeyUsage is not 0x01 (e.g. TL session key), this field should be elided.
6	SoftLifeSpan_Ratio	Unsigned8	The percentage of the HardLifeSpan beyond which a key update will be initiated.
7	Security_Level	Unsigned8	Security level specified in Table 35.
8	MasterKeyID	Unsigned8	Crypto Key Identifier for the Master key used for generating MIC.
9	Time_Stamp	Unsigned32	Time of creating this message in TAITimeRounded form. This argument is time portion of the nonce used for generating MIC to encrypt and protect this method call
10	MIC	OctetString	The integrity check using AES_CCM*. The MIC length is chosen from MIC-32, MIC-64 and MIC-128 with Master key security level assigned in

			join process.
<b>Output arguments</b>			
<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>	<b>Argument description</b>

MIC is generated by an AES-CCM\* operation with following parameters:

- Authentication part: Element 1-7
- Encryption part: none
- Key: Master key, which has Crypto Key Identifier = MasterKeyID
- Nonce: formed Table 57 structure with;
  - EUI-64: EUI-64 address of Security manager
  - Nominal TAI Time: Time Stamp field conveyed in Key\_Policy\_Update() request.

The SoftExpirationTime in the Key Descriptor is updated following a successful MIC check on the parameters of this method call. All the parameters shall be concatenated from the first element to the one before last element (thus excluding the integrity check). The key SoftLifeSpan\_Ratio is the percentage of the difference between the ValidNotAfter and the ValidNotBefore time. For example, a SoftLifeSpan\_Ratio of 50% would cause a key update half way between the ValidNotBefore and the ValidNotAfter.

#### 7.11.4 Device security management object alerts

Table 98 describes the DSMO alerts.

**Table 98 – DSMO Alerts**

<b>Standard object type name(s): Device security management object (DSMO)</b>					
<b>Standard object type identifier: 125</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Security alerts on the state of the communication</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority (Enumerated: urgent, high, med, low, journal)	Value data type	Description of value included with alert
0 = Event	2 = Security	0 = Security_DPDU_Fail_Rate_Exceeded	6 = Medium	Type: Unsigned16 Initial default value: 0 Valid value set: [0-65535]	Alert generated after the preconfigured DPDU failure rate threshold is exceeded. The value conveys the number of failures during the time period
0 = Event	2 = Security	1 = Security_TPDU_Fail_Rate_Exceeded	6 = Medium	Type: Unsigned16 Initial default value: 0 Valid value set: [0-65535]	Alert generated after the preconfigured TPDU failure rate threshold is exceeded. The value conveys the number of failures during the time period
0 = Event	2 = Security	2 = Security_Key_Update_Fail_Rate_Exceeded	6 = Medium	Type: Unsigned16 Initial default value: 0 Valid value set: [0-65535]	Alert generated after the preconfigured updating security key failure rate threshold is exceeded. The value conveys the number of failures during the time period

## 8 Physical layer

### 8.1 General

The physical layer (PhL) is responsible for converting the digital data information into, and from, radio frequency energy emitted, and captured, by a device's antenna. This clause also specifies the operating frequencies, transmission power levels, and modulation methods used. This standard uses an existing international standard (IEEE Std 802.15.4) as the default PhL (Type A field medium). Future versions of this standard may define alternate physical layers.

The PhL provides two services, the PhL data service and the PhL management service. These services are collectively known as the PhSAP. The PhL data service (PhD) enables the transmission and reception of actual user data (PPDUs) across the physical radio channel. The PhL management service is used to control the operating functions of the radio such as frequency selection, transmit power, etc.

PPDU in this standard is as defined in IEEE Std 802.15.4. Each PPDU consists of a synchronization header (SHR), a PHY header (PHR), and a PHY payload (PSDU). A start frame delimiter (SFD) in the SHR is commonly used as a timing signal.

As a certified IEEE Std 802.15.4-compliant radio, the device will be allowed to operate license-free in most countries around the world. The device vendor will be responsible for certifying its devices compliant with this standard and complying with any specific country variations.

### 8.2 Default physical layer

#### 8.2.1 General requirements

The PhL shall be based on the IEEE Std 802.15.4 standard, with additional requirements and exceptions specified herein.

The device vendor will be responsible for certifying its devices compliant with this standard and complying with any specific country variations.

#### 8.2.2 Additional requirements of IEEE Std 802.15.4

##### 8.2.2.1 Timing requirements

The IEEE Std 802.15.4 specification implicitly assumes that the operating channel will be static. This standard requires that the PhL support changing the frequency for every PPDU transmitted. Timing requirements are specified in Table 99.

**Table 99 – Timing requirements**

Event	Requirement
Time to change RF channels	< 200 µs
Time to switch from receive to transmit (with PA ON)	< 200 µs
Time to switch from transmit (with PA on) to receive	< 200 µs
Time for consecutive receptions	< 200 µs
Time from MAC request to first bit transmission	< 100 µs

##### 8.2.2.2 Carrier sense disable

The typical use of an IEEE Std 802.15.4 physical layer requires the use of a CSMA scheme to prevent collisions and increase coexistence. This scheme can delay transmission of a PPDU due to random back off times for channel acquisition.

This standard requires that this behavior be optional. Depending on system configuration, the PhL shall disable the CSMA operation as requested by the DL.

### **8.2.2.3 Number of channels**

The PhL shall support a minimum of 15 frequency channels in any configuration. IEEE Std 802.15.4 channels 11-25 shall be supported. Support of channel 26 as defined in IEEE Std 802.15.4 is optional.

NOTE Channel 26 is not mandatory due to commonly encountered regulatory constraints near the band edge.

### **8.2.2.4 Over-the-air data rate**

The PhL shall support a raw (over-the-air) data rate of 250 kbit/s.

### **8.2.2.5 Transmit power limits**

The PhL shall require the transmit power level to be a minimum of 0 dBm at full power. This power level shall be measured using procedures as defined by the regulatory requirements applicable to the environment in which the device will be deployed.

The PhL shall require the transmit power level to be adjustable from -5 dBm to maximum power of the device with 5 dB or smaller increments. The accuracy of each power level setting shall be  $\pm 4$  dB or better.

This standard does not specify the type or gain of antenna a device shall use.

The maximum radiated power level shall not exceed the regulatory requirements of the country(s) in which the device is used.

## **8.2.3 Exceptions to IEEE Std 802.15.4**

### **8.2.3.1 General**

This subclause lists the requirements of this standard that are deviations or omissions from the IEEE Std 802.15.4 specification.

### **8.2.3.2 Frequency bands**

Although the IEEE Std 802.15.4 standard supports multiple frequency bands, a device compliant with this standard shall operate in the license-exempt 2 400–2 483,5 MHz band only. This standard does not support the 868 and 902 MHz sub-bands.

## 9 Data link layer

### 9.1 General

#### 9.1.1 Overview

The data link layer (DL) in this standard is designed with the general goal of constraining the range of build options for a field device, while enabling flexible and innovative system solutions.

The DL specification provides a set of capabilities that are well defined and verifiable for each field device that participates in a DL subnet. The DL can be conceptualized as a table-driven state machine that operates independently on each field device. A DL subnet is a group of devices provided with a matched set of table-driven configurations by the system manager.

The DLs building blocks include timeslots, superframes, links, and graphs. The system manager may assemble these building blocks to configure a device in one of three general operational alternatives, slotted channel hopping, slow channel hopping, and hybrid slotted/slow hopping. (See 9.1.7.2 for a discussion of channel hopping.)

A timeslot is a single, non-repeating period of time. The timeslot durations in this standard are configurable to a fixed value such as 10 ms or 12 ms. Once a timeslot duration is selected, all timeslots generally have the same duration and they are re-aligned to a 4 Hz cycle at each 250 ms clock interval. (See 9.1.9 for a discussion of DL timekeeping.)

A superframe is a collection of timeslots repeating on a cyclic schedule. The number of timeslots in a given superframe determines how frequently each timeslot repeats, thus setting a communication cycle for devices that use the superframe. The superframe also has an associated reference channel hopping pattern. (See 9.1.8 for a discussion of superframes.)

Links are connections between devices. When the system manager defines paths between devices, the devices receive link assignments. A link assignment repeats on a cyclic schedule, through its connection to an underlying superframe. Each link refers to one timeslot or a group of timeslots within a superframe, its type (transmit and/or receive), information about the device's neighbor (the device on the other end of the link), a channel offset from the superframe's underlying hopping pattern, and transmit/receive alternatives.

This standard supports graph routing as well as source routing. A directed graph is a set of directed links that is used for routing DPDUs within a DL subnet. Each directed graph within the DL subnet is identified by a graph ID. In source routing, the originating device designates the hop-by-hop route that a DPDUs takes through a DL subnet. Graph routing and source routing may be mixed. (See 9.1.6 for a discussion of routing.)

#### 9.1.2 Coexistence strategies in the data link layer

This standard incorporates several strategies that are used simultaneously to optimize coexistence with other users of the 2.4 GHz radio spectrum, as described in 4.6.10. Most of these strategies are handled adaptively by the DL in conjunction with the system manager.

#### 9.1.3 Allocation of digital bandwidth

The DL is a table-driven state machine that provides prioritized access to digital bandwidth for directional communication among devices within a DL subnet. The state machine operates on one timeslot at a time.

Digital bandwidth is allocated by a system management function. For example, a field device may need to report every 10 s. A level of service is arranged through the system management function, to ensure that the digital bandwidth is available when needed. The system management function, in turn, arranges the digital bandwidth in the field device and the intermediate field routers.

A link is the basic unit of service within the DL. A link may be incoming, outgoing, or bidirectional. It may be unicast or broadcast (see 9.1.9.4.2).

DL digital bandwidth may be allocated to deliver an average level of service, for example, 10 DPDUs of available digital bandwidth (multi-hop) per minute. Alternatively, DL digital bandwidth may be allocated to support a reporting interval and a level of service, for example, one DPDU (including retries) every 15 s, with 2 s maximum latency to a backbone connection.

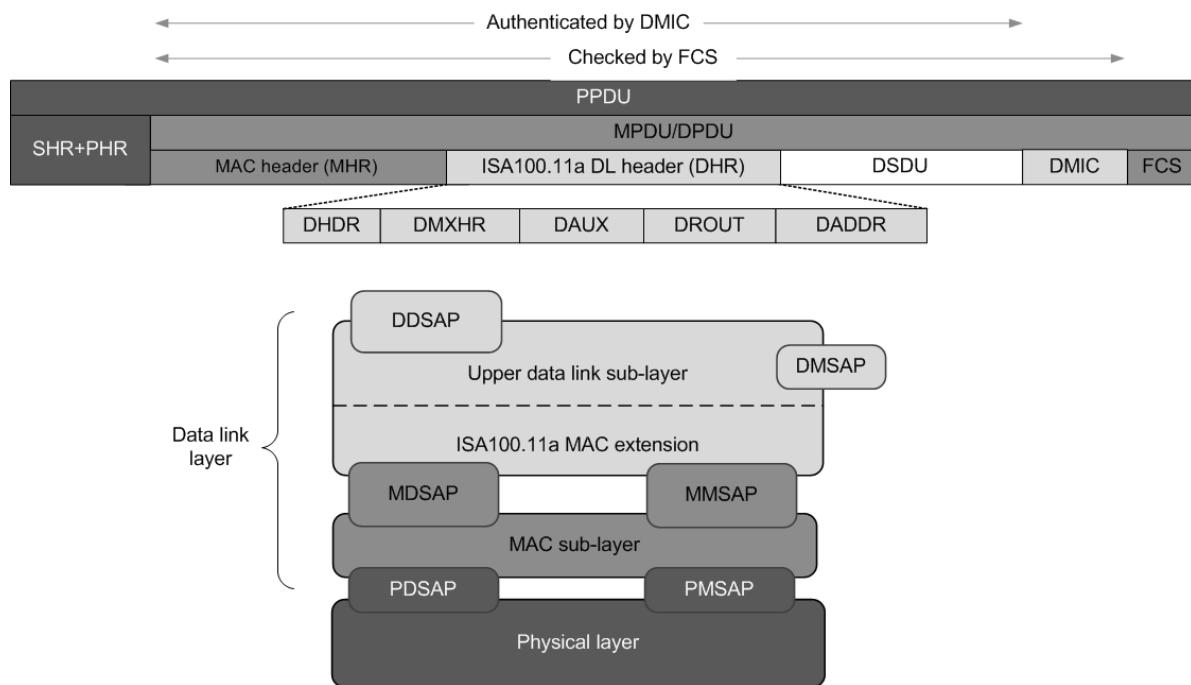
DL digital bandwidth may be organized as a pool that can be shared by a collection of devices using the corresponding links. A level of service may be delivered by ensuring that sufficient shared, contention-based capacity is available.

For more granular channel allocation, links may be tied to particular groups of DPDUs.

A service level may be delivered with a combination of specific link allocations and generally available shared digital bandwidth. For example, for each report, a dedicated link may be allocated for the first transmission to each of two neighbors, with retries using shared digital bandwidth.

#### 9.1.4 Structure of the data link layer

The general structure of the DL protocol data unit (DPDU) in this standard is shown in Figure 53.



**Figure 53 – DL protocol suite and PPDU/DPPDU structure**

The DL compliant with this standard includes:

- A subset of the IEEE Std 802.15.4 MAC, as described in 9.1.5. This handles the low-level mechanics of sending and receiving individual data frames. The SHR, PHR, MHR, and frame check sequence (FCS) of every DPPDU are as described and specified in IEEE Std 802.15.4.
- The extension to the MAC, including aspects of the DL that are not specified by IEEE but are logically MAC functions.
- The upper DL that handles link and mesh aspects above the MAC level.

Components of the DL header in this standard are described in 9.3.1.

#### 9.1.5 The data link layer and IEEE Std 802.15.4 media access control layer

This standard uses the IEEE Std 802.15.4 MAC (called IEEE MAC herein). IEEE MAC data frames are used as specified by IEEE Std 802.15.4. See 9.3.3 for detail.

The IEEE MAC describes various features that are not used by this standard's DL (called the DL herein). In summary, only IEEE MAC data frames are used by the DL.

A device compliant with this standard never associates with a coordinator in the sense defined by the IEEE MAC. None of the IEEE MAC functions involving FFDs are used by this standard.

Within the limited context of data frames, there are some features that are not supported by the IEEE MAC. These features are implemented via the MAC extension of this standard, which enhances the IEEE MAC with features that are logically MAC functions, but that are not currently included in the IEEE MAC.

The DL and IEEE MAC each specify an entity called a superframe (see 9.1.8), but the DL uses no aspects of the IEEE MAC superframe specification.

DL acknowledgements are used to convey time information for clock correction, in addition to providing acknowledgement. This feature is not available in the IEEE MAC acknowledgement. For this and other reasons, MPDUs are unacknowledged at the IEEE MAC level, with acknowledgements compliant with this standard being handled by this standard's MAC extension as IEEE data frames.

The IEEE MAC includes active and passive scans, which are not used in this standard. This standard has alternative active and passive scans, using IEEE data frames.

The IEEE MAC backoff and retry mechanism is not used by the DL. Instead, the DL implements its own retries, involving spatial diversity (retries to multiple devices), frequency diversity (retries on multiple radio channels), and time diversity (delaying the DPDU). The manner and degree of these elements of diversity are not fixed, but configured by the system manager. More generally, this standard's DL uses CSMA-CA, but the details are different from CSMA-CA as defined in the IEEE MAC. Various aspects of the IEEE MACs CSMA-CA behavior are not used, and CSMA-CA functions are handled in this standard's DL.

The standard includes three exceptions to the IEEE MAC.

- Acknowledgements and solicitations, which are technically data frames in IEEE, use a destination-addressing mode of 00 and a source-addressing mode of 00. In IEEE, this combination is limited to IEEE acknowledgements and IEEE beacons.
- Advertisements and certain duocast acknowledgements, which are technically data frames in IEEE, use a destination-addressing mode of 00 and a source-addressing mode of 10 (16-bit short address). In IEEE, this combination implies that the frame is directed to the PAN coordinator, which does not apply to this standard.
- Burst advertisements, which are technically data frames in IEEE, use an interframe spacing that is less than LIFS. LIFS is intended to protect the IEEE CSMA acknowledgement, which does not apply to operation in this standard. See 9.1.13.3.

## 9.1.6 Routes and graphs

### 9.1.6.1 General

Routes are configured by the system manager, based on reports from devices that indicate instantaneous and historical quality of wireless connectivity to their immediate neighbors. The system manager accumulates these reports of signal quality to make routing decisions. The signal quality reports are standardized, but the routing decision process within the system manager is not standardized. Once the system manager makes its routing decisions, it uses standard DPDUs to configure routes within each device in the DL subnet. (See 9.1.13 and 9.1.14 for a review of neighbor discovery.)

DL routing is adaptive at two levels:

- Field devices make instantaneous adaptive forwarding decisions. Field devices are normally configured with path diversity, so that if one link fails somewhere along the route, the device can immediately send the DPDU along an alternative path.
- If, over time, certain links have consistent connectivity issues, this is reported to the system manager, which can then reconfigure the field device to use different links.

Within each DPDU, DL routing instructions are placed in the DL's DROUT sub-header (see 9.3.3.6). When a DPDU is addressed to an immediate neighbor, such as during the network join process, the route is simply the address of that neighbor. When the DPDU is being sent to a more distant device, a single graph number indicates how the DPDU flows through the DL subnet to reach that address. These two approaches may be combined. For example, a DROUT sub-header may contain two entries, the first identifying an immediate neighbor for the first hop, and the second indicating a graph that is used for the rest of the route through the DL.

Routes that identify specific addresses, also known as source routing, are not as adaptive as routes based on graphs. When a route is based on a series of addresses, each device along the route becomes a single point of failure. Graphs, on the other hand, should be configured with multiple branches at each hop, so that if there is a connectivity problem with one neighbor, the device can immediately forward the DPDU to a different neighbor.

Source routing is useful for quick, transitory communications between devices, such as during the join process. Source routing may also be used when graph route resources are scarce.

The DROUT sub-header is constructed by the DL at the point where the DPDU enters the DL from the network layer. Routes are selected by table lookup based on contract ID, destination address, or by default. Once a route is selected the DPDU follows the route until it arrives at the DL termination point, which may be its ultimate destination, or alternatively may be a waypoint along the route, such as a backbone router or the system manager. At the DL termination point, the DPDU is passed up to the network layer.

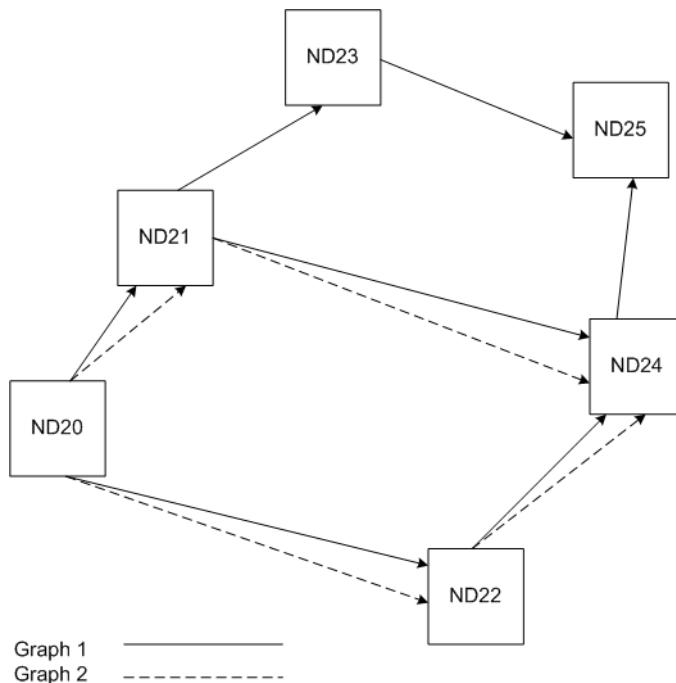
The DROUT sub-header includes a forwarding limit field which is used to limit the number of times that a DPDU can be forwarded in a DL subnet. The forwarding limit is initialized by the DL when the route is assigned and decremented with each hop until it reaches zero, triggering discard of the DPDU on the subsequent hop.

### 9.1.6.2 Graph routing

A graph is a set of directed links that is used for routing messages within a DL subnet. Each graph designated by the system manager for routing within a DL subnet is identified by a graph ID.

The links associated with each graph are configured by the system manager. A DL subnet may have multiple graphs, some of which may overlap. Each device may have multiple graphs going through it, even to the same neighbors.

Figure 54 illustrates an example of graph routing.

**Figure 54 – Graph routing example**

In Figure 54, ND20 communicates with ND25 using graph #1. To send a DPDUs on that graph, ND20 may forward it to ND21 or ND22. From those devices, the DPDUs may take several alternate routes, but either way, following graph #1, the DPDUs will arrive at ND25. Similarly, to communicate with ND24, ND20 may send DPDUs on graph #2 through ND21 or ND22, either of which in turn will forward the DPDUs to ND24.

Figure 54 shows all graphs originating from ND20, but the same graphs may be used by any node. For example, the system manager may configure ND21 to use Graph 1 for its communication with ND25.

NOTE 1 The header of the DPDUs may be changed as it moves through the network.

Table 100 and Table 101 reflect the contents of graph tables on ND20 and ND21. These graph tables roughly correspond to data structures within each device for the topology shown in Figure 31. For example, a DPDUs following Graph 2 will look up Graph ID 2 in each router along the route to find out which neighbors it can use for the next hop.

**Table 100 – Graph table on ND20**

Graph ID	Neighbor address
1	21, 22
2	21, 22

**Table 101 – Graph table on ND21**

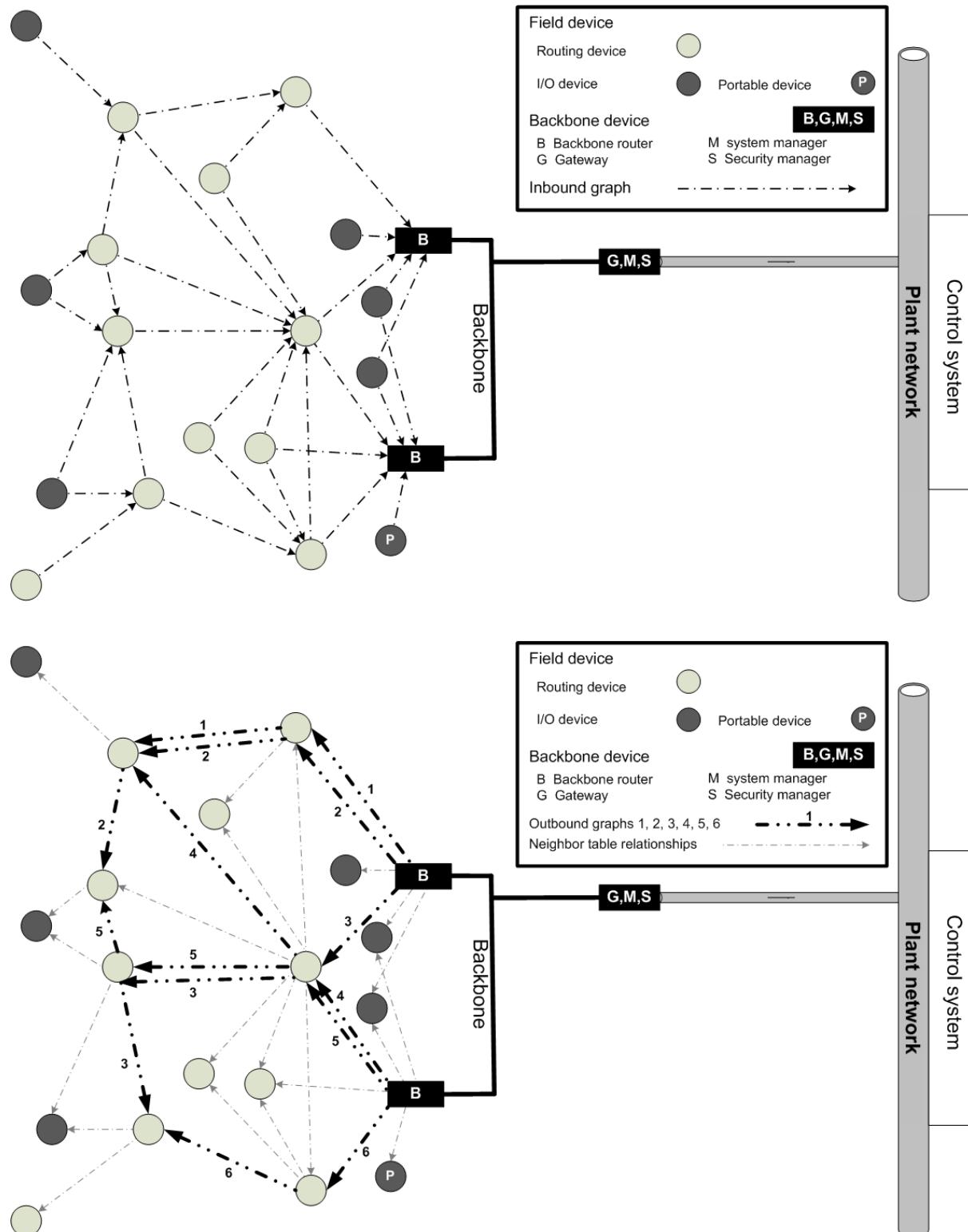
Graph ID	Neighbor address
1	23, 24
2	24

Each graph within a DL subnet is identified by a graph ID. A DL message typically originates in the DL subnet at a field device, a gateway/manager, or a backbone router. To send a message on a graph, the originating DL device includes a graph ID in the DPDUs DROUT sub-header. The DPDUs travels along the paths corresponding to the graph ID until it reaches its destination or is discarded.

In order to route DPDU over a graph, each device along the path needs to maintain a graph table containing entries that include the graph ID and neighbor device address. A device routing a DPDU performs a lookup based on graph ID and sends the DPDU to any one of the applicable neighbors. Once a neighbor acknowledges receipt of the DPDU, the device releases it from a DL message buffer.

Diverse graph paths (a branch) may be established by configuring more than one neighbor associated with the same graph index. A branch may be configured with a preferred neighbor, indicating that the DL should attempt to transmit the DPDU to the preferred neighbor first, even if there is an earlier opportunity to transmit to other neighbors. If no preferred neighbor is designated, the DL should treat all branches equally, transmitting the DPDU at the first opportunity that presents itself. If the first transmission does not result in DL-layer acknowledgement, the DL is normally configured to use alternative branches for retries.

Figure 55 provides examples of routing graphs that are inbound (toward the backbone) and outbound (away from the backbone). The basic organization of inbound and outbound routing graphs may be very similar to each other, but pointing in opposite directions, as shown in Figure 55. The system manager configures routing relationships among devices in a DL subnet.



**Figure 55 – Inbound and outbound graphs**

The top half of Figure 55 shows an inbound graph in an example DL routing configuration. An inbound graph enables a set of field devices to send DPDUs toward the backbone or system manager. A single graph may be used for inbound routing in a small DL subnet, as shown in the top half of Figure 55. It is possible and often desirable for the system manager to define multiple inbound graphs, particularly as the network scales. As shown in the top half of Figure 55, if field devices have multiple neighboring routers, then diversity is typically inherent in the inbound graph.

**NOTE 2** The illustrative inbound graph in Figure 55 can be problematic for fragmented NPDUs, because the fragments may arrive at different backbone routers and, in that case, cannot be reassembled. For device contracts supporting payload sizes that do not fit in a single DSDU, graphs directed toward the backbone should be arranged so that defragmentation can occur within the scope of a single backbone router. For most contracts covering communication toward the backbone, fragmentation is not needed and therefore need not be covered by the corresponding contract, as indicated by the contract's maximum supported payload size. Contracts involving fragmentation may, in some configurations, involve use of specific graphs with reduced path diversity to ensure that a set of fragmented NPDUs are all delivered to the same backbone router. Applications for fragmentation toward the backbone include download of waveforms from a device to a gateway, with fragmentation enabling more efficient operation of a UDO.

The bottom half of Figure 55 shows a set of outbound graphs in an example routing configuration. An outbound graph is typically used to send DPDUs from backbone devices to field devices. As shown in the bottom half of Figure 55, multiple graphs may be used for outbound routing, with each outbound graph corresponding to a group of devices within radio range of the graph.

In this example, inbound DL graph routing ends at the backbone, at which point the NL takes over routing responsibilities. The relationship between the DL and the backbone is described in 5.5.

Although all of the examples above show inbound and outbound graphs, these are not actually different types of graphs. They are just graphs that happen to point in opposite directions. Peer-to-peer routing is also supported by the standard. The system manager may arrange a graph to follow any route where connectivity exists.

A DSDU is forwarded along a graph until the graph is terminated. If the DPDUs destination address matches the device's address, then the DSDU has reached its destination and the graph is terminated. Alternatively, if the graph number in the DPDUs does not have a corresponding entry in the lookup table `dlmo.Graph` (see 9.4.3.6), the graph has reached its termination point.

### 9.1.6.3 Graph extensions

The bottom half of Figure 55 shows that outbound graphs do not necessarily extend to all devices on the periphery of a DL subnet. Nonetheless, such devices are covered by the outbound graphs implicitly, through a graph extension mechanism. A DL automatically extends graphs by checking the DPDUs destination address for a neighbor table entry, thus indicating that the DL destination is one hop away. If it is, the router treats the neighbor as if it were listed in the graph, thereby extending the graph for that DPDUs. More formally, when the DPDUs destination address is in a DL's neighbor table, and the DPDUs is being routed with a graph, the DL shall treat that graph as including that neighbor for the purpose of routing that DPDUs even if the graph does not explicitly refer to the neighbor. All routers shall support this basic form of implicit graph extensions.

An explicit graph extension field in the neighbor table provides an additional degree of control. If a graph is specifically designated in a neighbor table, as described in 9.4.3.4.2, the neighbor is not only treated as being covered by the graph; the neighbor is also given preferential treatment. If the neighbor is designated as the graph's last hop, a DPDUs following that graph shall be forwarded exclusively to that neighbor. If the neighbor is designated as a preferred branch, the DL should attempt to forward an applicable DPDUs to that neighbor before other neighbors.

Support for the explicit graph extension field in the neighbor is optional in a device, and the presence or absence of this capability is reported to the system manager, through `dlmo.DeviceCapability`, when the device joins the network. All routers support the basic implicit graph extension capability, but only certain routers support the explicit last hop and preferred branch indicators in the neighbor table.

### 9.1.6.4 Source routing

Source routing is a general method of routing supported by this standard. In source routing, the originating device may be configured to designate a hop-by-hop route for a DPDUs to follow through a DL subnet. A simple use of source routing is a DPDUs directed one hop away to a specific neighbor, such as for joining. When a source-routed DPDUs arrives at an

intermediate device, the intermediate device examines the path information in the DPDU to determine the neighbor to which it should forward the DPDU.

A source route is a list of entries specifying the route that a DPDU shall follow through the DL subnet. The first entry in the list specifies the next hop, and the list is shortened as the DPDU moves through the DL subnet. Source routing entries can specify graphs or addresses, thus allowing graphs to be chained. 12-bit graph numbers within a source route shall be encoded as 1010ggggggggggg.

The DPDU header compresses a source route to a single octet in the common case where a single graph route is specified and the graph number is <=255 (encoded as 10100000gggggggg). This is commonly referred to as graph routing, but formally it is a source route containing a single graph.

In the provisioning or joining process, an EUI-64 is used to address a device that has not yet received a network address. This case is encoded as a route with a graph of zero and a DADDR address of zero (see 9.3.3.6 and 9.3.3.7), indicating that the EUI-64 can be found in the MAC header.

When a DPDU is received by the DL through its wireless link, the following processing steps shall be followed, in order, to determine whether the DL route has terminated and to update the source route in the DPDU header:

- The DADDR sub-headers destination address is checked to see if it matches the address of the receiving device. If there is a match, the DSDU has reached its final destination and the DSDU shall be passed to the network layer as described in 9.2.4. (The DADDR destination address is encoded as zero, in the case where the destination address is duplicated in the MHR. See 9.3.3.7. In that case, the match is indirect, based on the MHR destination address.)
- The first entry in the source route is deleted if appropriate, thus shortening the source route by shifting the second and subsequent entries (if any) into the prior positions (shift left). The first entry shall be deleted unless it is a graph number that has not reached its termination point.
- If the route has no remaining entries, the route has terminated and the DSDU shall be passed up to the network layer as described in 9.2.4.

If the DSDU is not passed to the NL, the DPDU shall be discarded if the forwarding limit (in the DROUT sub-header) is zero. If the forwarding limit is positive, the forwarding limit shall be decremented and the DPDU placed on the message queue for forwarding.

When a DSDU is intended to be routed through the backbone, the DL route should terminate at the backbone router. If a route does not terminate in backbone router, it is forwarded by the DL and never processed by the backbone routers network layer, thus allowing peer-to-peer messaging to occur within the DL subnet through a backbone router.

The routing methods are examples of different ways to configure the DL routing capability that is resident in all field routers compliant with this standard. The system manager shall configure all graphs within a DL subnet. The ability to configure routing in any of several ways such as graph routing and/or source routing enables device interoperability.

#### **9.1.6.5 Route selection**

The route for a message through the DL subnet is selected when the message enters the DL (see 9.2.2). The route is stored in the DROUT sub-header for the reference of other devices that will route the DPDU. The initial selection of a route is based on decision rules in the DL. The following list shows the route selection criteria in order, whereby the route shall be selected based on the first condition that applies:

- The message has a 64-bit destination. A 64-bit destination address is used only during the join process, when a router is sending a response to an immediate neighbor that has not yet received a 16-bit address from the system manager. In that case, use the 64-bit address in the IEEE MAC, and use Graph ID=0 in the route.

- The ContractID is associated with a particular route. This may be used when a particular graph or source route is intended to provide a defined level of service.
- The 16-bit network destination address is associated with a particular route.
- The 16-bit network destination address is an immediate neighbor, such as during the join process.
- Otherwise, use the default route. Normally, the default will direct the message to the nearest backbone router, or to the system manager if there is no backbone router.

A single route may be designated as the default by the system manager, by designating a particular route as a default. The default route is typically configured to route messages to the system manager, or to a backbone router if there is no system manager on the DL subnet. A default route may sensibly be configured in conjunction with the establishment of a device's contract with the system manager. Additional routes may be configured as needed, such as to provide enhanced quality of service or to route messages to a peer device on the DL subnet.

### **9.1.7 Slotted hopping, slow hopping, and timeslots**

#### **9.1.7.1 General**

Three general operational alternatives are supported by the DL:

- Slotted channel hopping;
- Slow channel hopping; and
- Hybrid combinations of slotted and slow hopping.

These three operational alternatives are different ways for a system manager to configure a slotted hopping capability that is supported by every device in a DL subnet. Channel hopping schedules are configured by the system manager through advertisements and the dlmo.Superframe attribute.

Slotted and slow hopping provide different ways to configure series of timeslots. The system manager determines the mode of operation and assigns the use of superframes, which are cyclic collections of timeslots. (See 9.1.8 for further discussion of superframes.) This provides flexibility and interoperability without requiring excessive complexity within the devices.

From the perspective of a field device, the DL can be visualized as a player piano with several keys. Each key corresponds to a DL transaction during a timeslot. There is a key for sending a pending message from the outbound queue, a key for listening for an incoming message, and so forth. The system manager provides a piano roll for the field device to play over and over again. The style may be slow hopping, slotted hopping, or a hybrid of the two. The field device does not know the difference; it simply mechanically plays each key at an exact time based on the instructions on the piano roll.

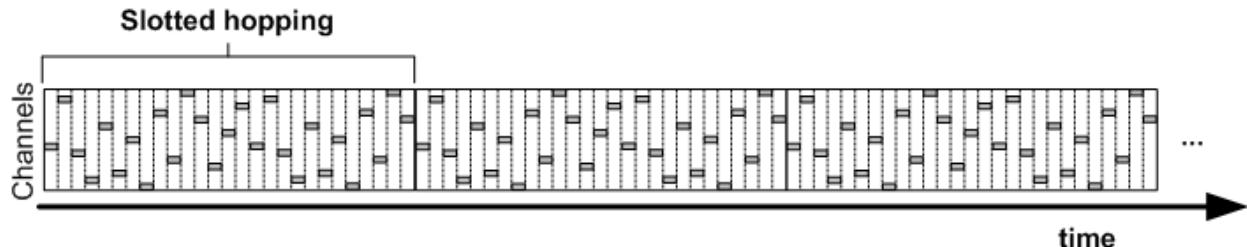
The rhythms within a timeslot are configurable, using timeslot templates provided by the system manager. There is a constrained series of operations that can be performed within a timeslot – transmit, listen, wait, timeout, and acknowledge – but those simple building blocks may be assembled with different timings. This is left flexible, under the control of the system manager.

Timeslot duration in a DL subnet is set to a specific value by the system manager when a device joins the network. Timeslot duration of 10-12 ms is intended to be typical. Timeslot duration is configurable to enable:

- Optimized coexistence with other systems, such as other DL subnets and WirelessHART;
- Longer timeslots to accommodate extended message wait times;
- Shorter timeslots to take full advantage of optimized implementations;
- Longer timeslots to accommodate serial acknowledgement from multiple devices (e.g., duocast);
- Longer timeslots to accommodate CSMA at the start of a timeslot (e.g., for prioritized access to shared timeslots);

- Longer timeslots to accommodate slow-hopping periods of extended duration;
- Timeslots to be synchronized with other non-standard-compliant networks to facilitate inter-routing.

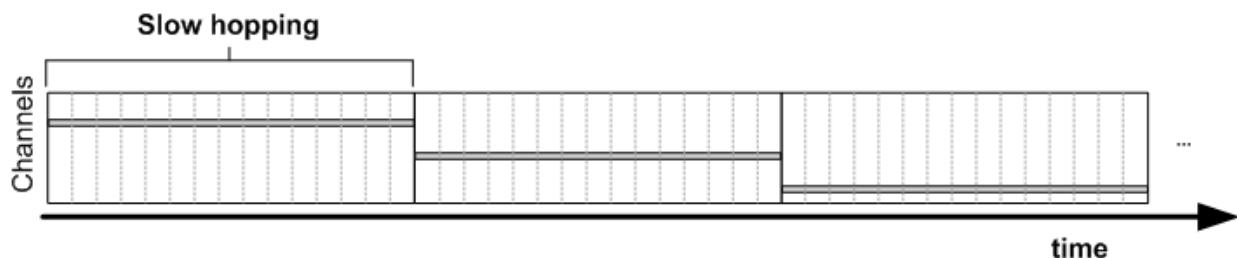
Figure 56 illustrates slotted hopping operation.



**Figure 56 – Slotted hopping**

In slotted hopping, channel-hopping timeslots of equal duration are used. Each timeslot uses a different radio channel in a hopping pattern. In slotted hopping, each timeslot is intended to accommodate a single transaction including one DPDUs and its acknowledgement(s).

Figure 57 illustrates slow hopping operation.



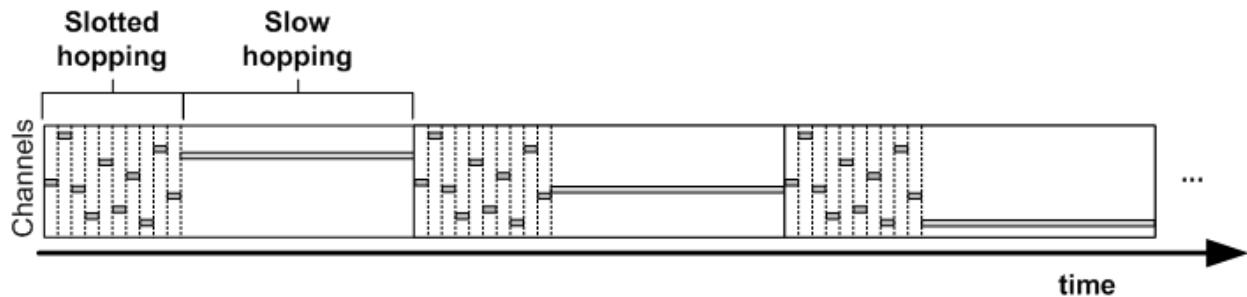
**Figure 57 – Slow hopping**

In slow hopping, a collection of contiguous timeslots is grouped on a single radio channel. Each collection of timeslots is treated as a single slow hopping period; however, as shown in Figure 57, timeslots still underlie slow hopping. Slow-hopping periods are configurable, typically on a scale of about 100 ms–400 ms per hop. Longer slow-hopping periods, potentially multiple seconds in duration, may be used to support devices with imprecise timekeeping and/or devices that have temporarily lost contact with the network. To enable device interoperability, all devices compliant with this standard shall support configuration of timeslot duration, as designated by the system manager.

NOTE In some regulatory domains, slow hopping of an IEEE Std 802.15.4 2.4 GHz radio is not permitted to exceed 400 ms per hop. However, in other regulatory scenarios there is no such constraint. The slow hopping period is set by the system manager, not by the standard; therefore a 400 ms constraint is not specifically enforced by this standard.

The structure, duration, and assignment of timeslots in slow-hopping periods are configured by the system manager, which determines the mode of operation and assigns the use of timeslots. This provides flexibility without requiring excessive complexity within the devices, enabling device interoperability.

Figure 58 illustrates a hybrid mode of operation.



**Figure 58 – Hybrid operation**

Hybrid operation uses slotted hopping and slow-hopping periods in a configured combination. For example, in Figure 58, a number of timeslots using slotted hopping is followed by a slow hopping period.

### 9.1.7.2 Channel hopping

#### 9.1.7.2.1 General

DL communications are intended to be distributed across multiple radio channels. (This standard, based on IEEE Std 802.15.4 at 2.4 GHz, uses the sixteen available channels 11-26.) The system uses channel hopping patterns which provide a specific sequence of channels for communication among collections of devices. Channel hopping begins at a designated offset in the hopping pattern, continuing through the pattern in order until the end, and then repeats the pattern.

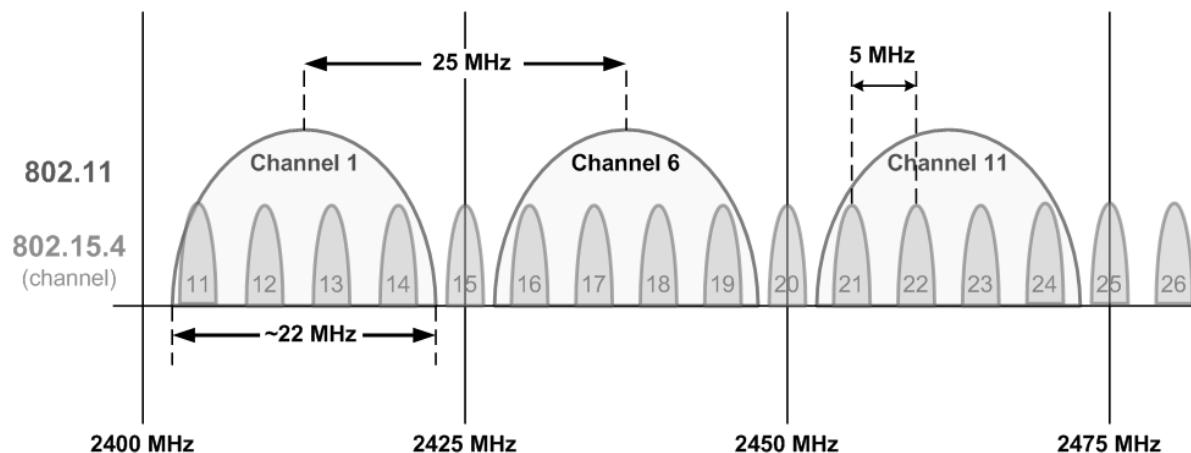
NOTE 1 IEEE Std 802.15.4 channels 11-26 are mapped to 0-15 in this standard. This overview refers to them as channels 11-26.

NOTE 2 This standard is based on devices that use one channel at a time. Multiple radios may be co-packaged by operating multiple instances of the DL simultaneously on different channels. Such multi-channel operation is not specified by this standard, but is not intentionally prohibited.

#### 9.1.7.2.2 Radio spectrum considerations (INFORMATIVE)

For radio communication, this standard uses IEEE Std 802.15.4 channels in the 2.4 GHz band. The IEEE Std 802.15.4 physical layer (2.4 GHz, DSSS) includes sixteen channels, numbered 11 through 26.

Figure 59 illustrates the 16 IEEE Std 802.15.4 channels along with three commonly used IEEE 802.11 channels.



**Figure 59 – Radio spectrum usage**

In Figure 59, the narrow channels 11-26 are IEEE Std 802.15.4 channels; these channels are substantially non-overlapping. Also shown are the wider IEEE 802.11 channels 1, 6, and 11; these three channels are common choices for IEEE 802.11 communication.

As Figure 59 shows, IEEE Std 802.15.4 channels 15, 20, and 25 do not substantially overlap any of the three common IEEE 802.11 channels. Therefore, IEEE Std 802.15.4 channels 15, 20, and 25 may reasonably be designated as slow hopping channels. These slow hopping channels may be configured for purposes such as neighbor discovery. For example, information about the network may be advertised by field routers on pre-designated slow hopping channels, so that any device seeking to join the network can limit its active and passive scans to these channels when discovering nearby field routers.

This illustration demonstrates how spectrum management techniques supported by the standard may be used to account for a commonly encountered scenario. Alternative configurations of WiFi, other users of the radio spectrum, and local regulatory restrictions may justify alternative configurations in practice.

Most DL communications are intended to be distributed across all available IEEE Std 802.15.4 channels (11-26). The standard defines channel hopping patterns, each providing a specific sequence of channels to use. Channel hopping patterns for this standard are selected to have certain properties intended that minimize the occurrence of unmanaged and repeated collisions with co-located wireless devices, particularly IEEE 802.11 devices.

#### **9.1.7.2.3 Channel 26 and other blocked channels**

Support for channel 26 is optional in the standard, because some implementations may encounter regulatory restrictions at the upper band edge. In addition, a device may be blocked from using other channels, but only if necessary for regulatory compliance. A list of channels that the device supports is reported to the system manager when the device joins the network, through the attribute `dlmo.DeviceCapability.ChannelMap`. A device may be configured with links that use such unsupported channels; in which case the device shall treat those links as idle in timeslots where the unsupported channels are used.

Since support for channel 26 is optional in the standard, a system manager may sensibly limit DL subnet operation to channels 11-25. Hop patterns in the standard include channel 26, but these hop sequences may be shortened by excluding channel 26 from the channel map in each superframe.

#### **9.1.7.2.4 Spectrum management and selective channel utilization**

Multiple methods are available for limiting use of busy or undesirable radio channels, including clear channel assessment (CCA), spectrum management, and selective channel utilization.

Timeslots are normally configured to check for a clear channel before transmitting, using the CCA mechanism defined in the IEEE802.15.4 standard. CCA causes a device that is about to initiate transmission to relinquish a timeslot if interference is detected prior to transmission. See 9.1.9.4.3 and 9.1.9.4.8.

Spectrum management is a form of selective channel utilization. Spectrum management limits the DL configuration to a subset of channels. Limiting slow hopping to channels 15, 20 and 25 is an example of spectrum management. Another example is when a system manager blocks certain radio channels that are not working well or are prohibited by local policy. Spectrum management is handled by the system manager, through the way that it configures a device. See 9.1.8.4.7.

Additionally, a device may autonomously treat transmit links on problematic channels as idle, thus reducing unnecessary interference and wasted energy on channels with a history of poor connectivity. A device skipping links in this manner should periodically test the links to verify that they remain problematic. Such selective channel utilization can be disabled by the system manager on a link-by-link basis, through the attribute `dlmo.Link[].Type.SelectiveAllowed`. See 9.4.3.7.2, Table 182.

#### **9.1.7.2.5 Hopping patterns**

This standard supports five pre-programmed default hopping patterns, which shall be supported in every field device:

- Index 1: 19, 12, 20, 24, 16, 23, 18, 25, 14, 21, 11, 15, 22, 17, 13, (, 26)

- Index 2: Index 1 in reverse
- Index 3: 15, 20, 25 (intended for slow hopping channels)
- Index 4: 25, 20, 15 (Index 3 in reverse)
- Index 5: 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25 (, 26)

NOTE 1 Channel 26 is shown in parentheses, as it is supported by the standard but is not necessarily used due to commonly encountered regulatory constraints at the band edge. Graphics that follow mostly include channel 26, even though its use is commonly masked out by the superframe that uses the hop sequence.

NOTE 2 Channels are actually encoded within the DL to fall in a range of 0 to 15, as described in 09.4.3.2. For tutorial purposes, hopping patterns are expressed as their IEEE Std 802.15.4 channel numbers.

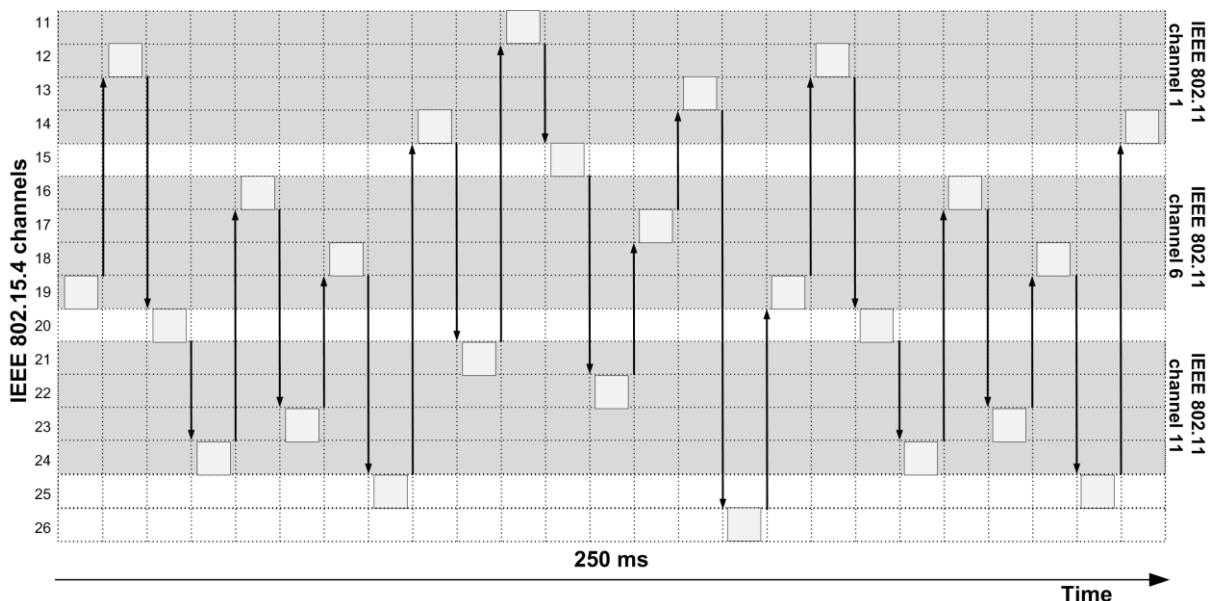
NOTE 3 Index 5 is intended to facilitate coexistence with WirelessHART.

The system manager can configure a device to use any of these hopping patterns for slotted hopping, slow hopping, or a hybrid.

Any channel or set of channels in a hopping pattern may be disabled (masked out) by configuration of the superframe that uses the hopping pattern, with the effect of shortening the hopping pattern for spectrum management.

Default hopping patterns for this standard are selected to have certain properties, whether channel 26 is included or not. Specifically, successive channels in most default hopping patterns are separated by at least 15 MHz or three channels. This property mitigates the effects of interference and multipath fading in industrial environments.

As shown in the example in Figure 60, default hopping pattern 1 is arranged so that consecutive hops do not overlap the same IEEE 802.11 channel.

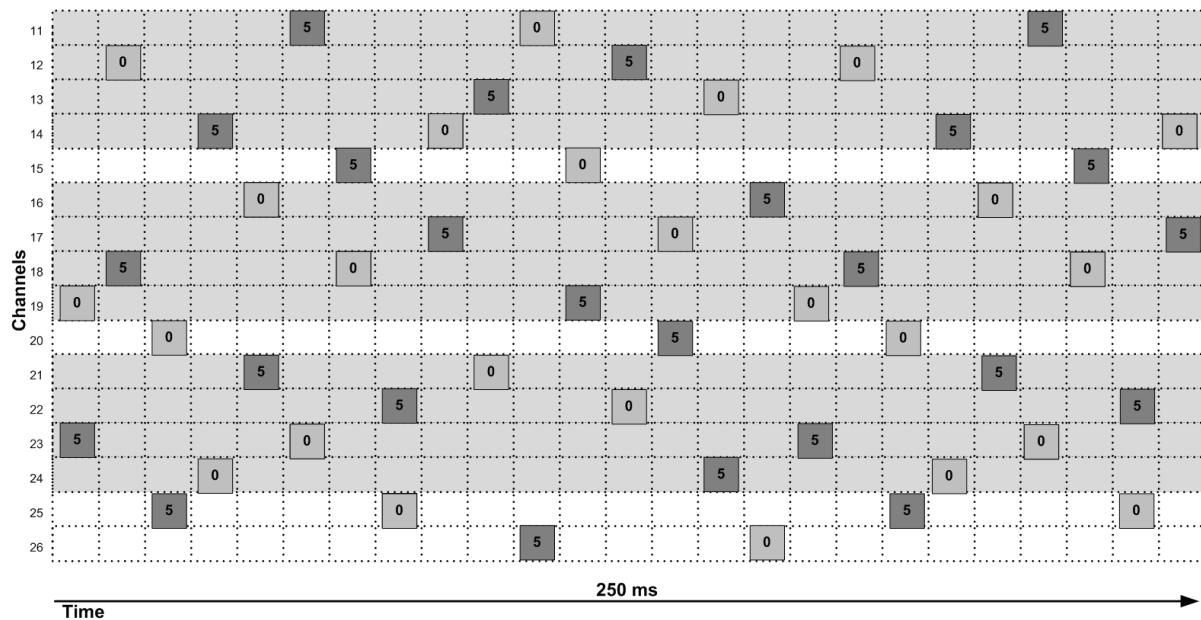


**Figure 60 – Default hopping pattern 1**

At least three channels separate each consecutive hop in pattern 1, resulting in frequency shifts of at least 20 MHz. When retries occur in consecutive hops, they will not encounter or cause interference in the same IEEE 802.11 channel.

For different groups of devices, it is desirable for devices to hop on non-interfering patterns.

Each hopping pattern is combined with a hopping pattern offset. If the hopping pattern offset is zero, then a baseline hopping pattern is used. If the hopping pattern offset is 5, then an offset of five is added to the baseline hopping pattern. Figure 61 shows how two groups of devices with different hopping pattern offsets into hopping pattern 1 may be used together without competing for the same radio channel at the same time.



**Figure 61 – Two groups of devices with different hopping pattern offsets**

NOTE 4 Superframe channel hopping offset is determined indirectly from the `dlmo.Superframe[].ChBirth` (`ChBirth`) attribute, which gives the starting reference of the channel hopping pattern following TAI time zero. This provides a baseline channel hopping sequence. Offsets from a baseline channel hopping sequence, given in the `dlmo.Link[].ChOffset` (`ChOffset`) attribute, are as described here and shown in Figure 61. While the same result can be achieved using `ChBirth` vs. `ChOffset`, care is needed because the two attributes are essentially reversed. Details of offset calculations can be found in 9.4.3.5.3.

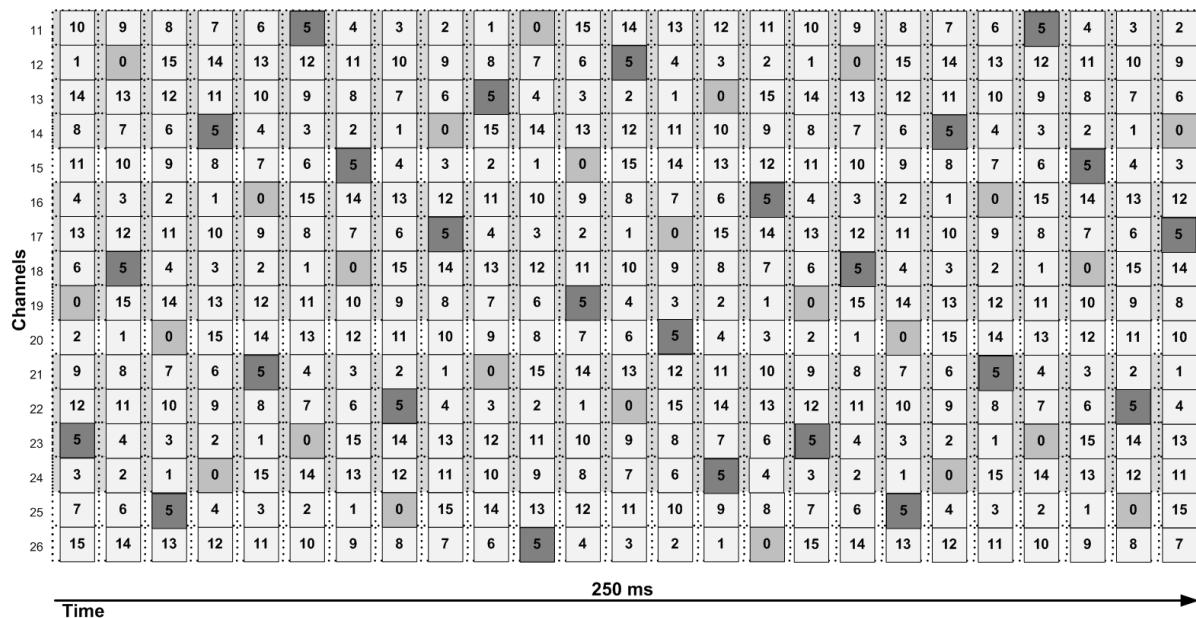
In Figure 61, boxes numbered 0 represent a group of devices using default hopping pattern 1; its hopping pattern is:

19, 12, 20, 24, 16, 23, 18, 25, 14, 21, 11, 15, 22, 17, 13, 26

Boxes numbered 5 in Figure 61 represent another group of devices using hopping pattern 1 with a hopping pattern offset of 5. A hopping pattern offset of 5 has the effect of essentially rotating the hop sequence to the left by 5, resulting in a hop sequence of:

23, 18, 25, 14, 21, 11, 15, 22, 17, 13, 26, 19, 12, 20, 24, 16

Figure 62 extends this principle, illustrating how different hopping pattern offsets may be used for a larger number of devices.



**Figure 62 – Interleaved hopping pattern 1 with 16 different hopping pattern offsets**

In Figure 62, 16 channels are available. Therefore, up to sixteen devices may use hopping pattern 1, each with a different hopping pattern offset (0-15).

As illustrated in Figure 62, a given hopping pattern may be interleaved by applying different hopping pattern offsets to various devices, superframes (see 9.1.8), or groups of devices. As a simple example, each of two device clusters may use the same hopping pattern with a different hopping pattern offset, so that the two clusters can share the radio spectrum without cross-interference. The clusters may be in the same DL subnet or in different DL subnets; as long as they accurately share a consistent timeslot duration and synchronized sense of time, their hopping patterns can be interleaved as shown in Figure 62.

The five default hopping patterns shall be resident in every field device compliant with this standard. Thus, routers can advertise concisely the hopping pattern that is being used and the current hopping pattern offset. This standard also supports customized hopping patterns in every field device, in addition to the seven defaults, so that the system manager can configure additional hopping patterns.

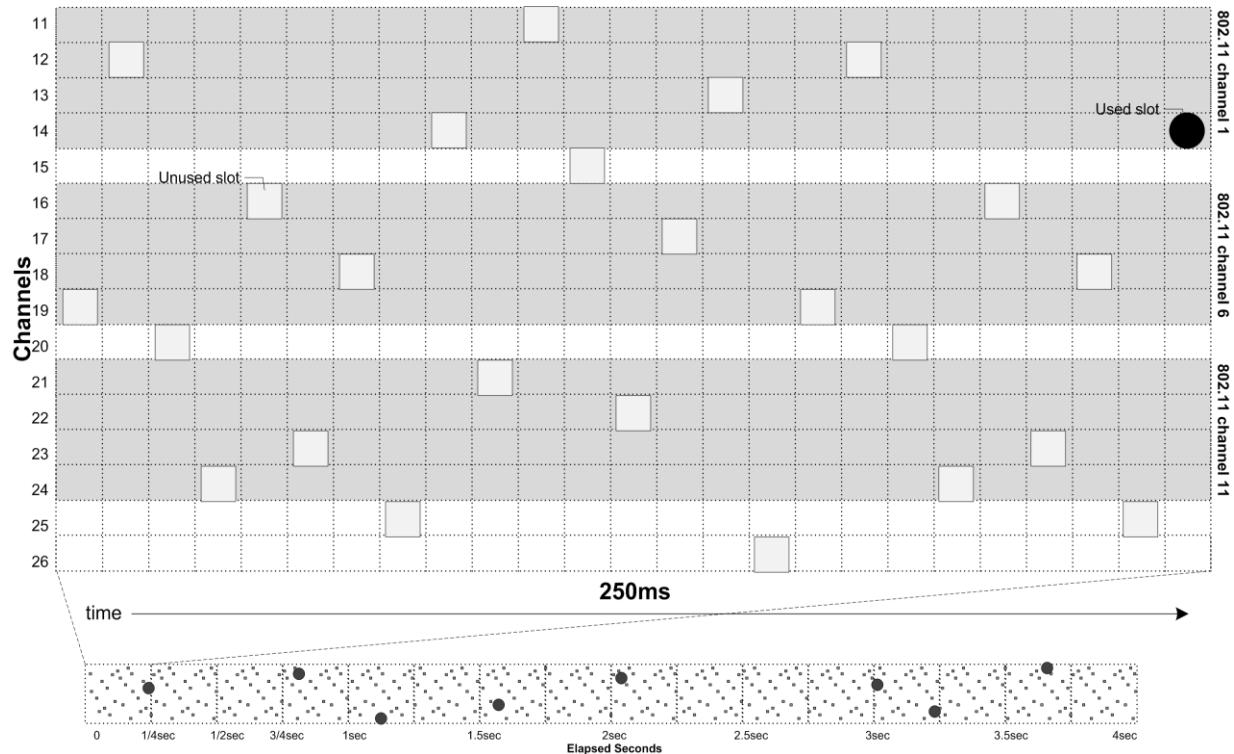
Each default hopping pattern is the same length as the number of channels being used. Thus, for example, hopping pattern 1 uses 16 channels and is 16 hops long. This property allows the hopping pattern to be interleaved as shown in Figure 62.

NOTE 5 Recognizing that some devices might not support channel 26, which is optional, systems may limit operation to 15 channels (11-25) with essentially the same result as described in this subclause.

#### 9.1.7.2.6 Timeslots and channel use

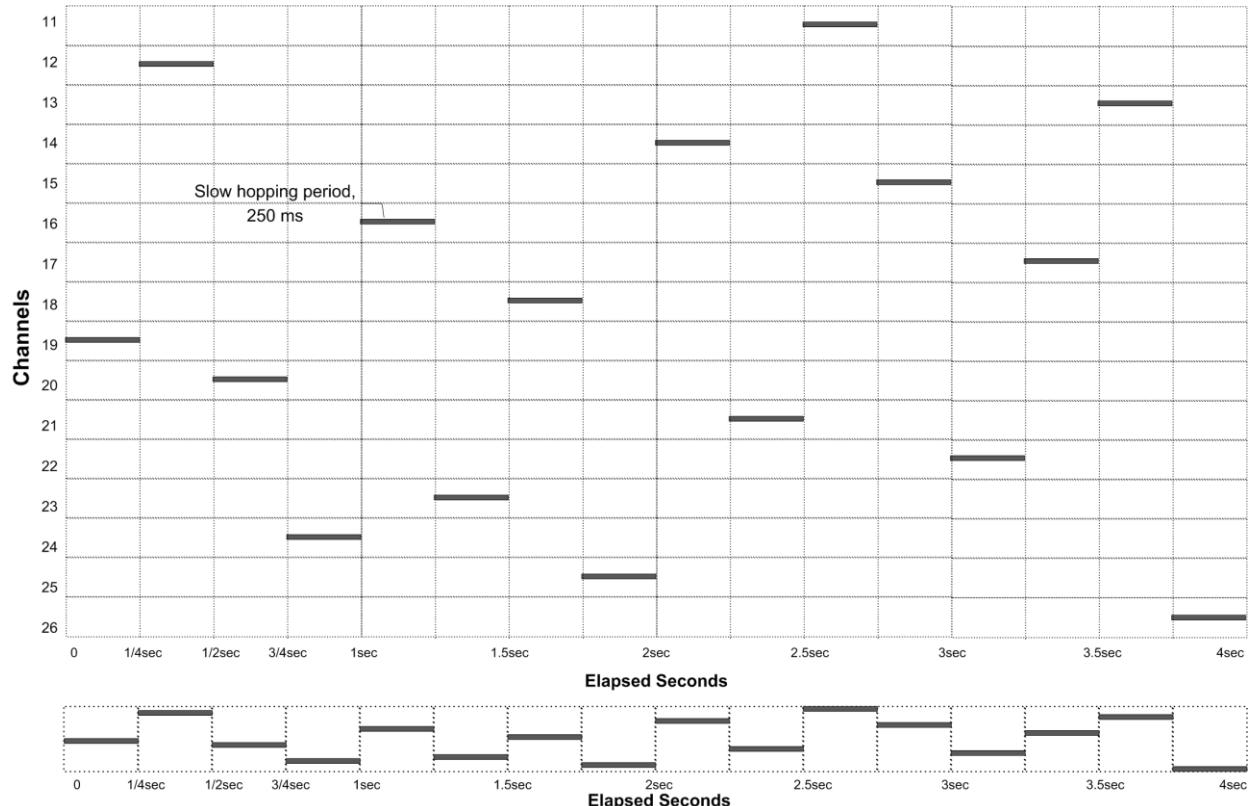
A system may use slotted hopping, slow hopping, or a hybrid combination of the two.

Figure 63 illustrates the use of slotted hopping. Each timeslot uses the next successive channel in the hopping pattern.

**Figure 63 – Slotted hopping**

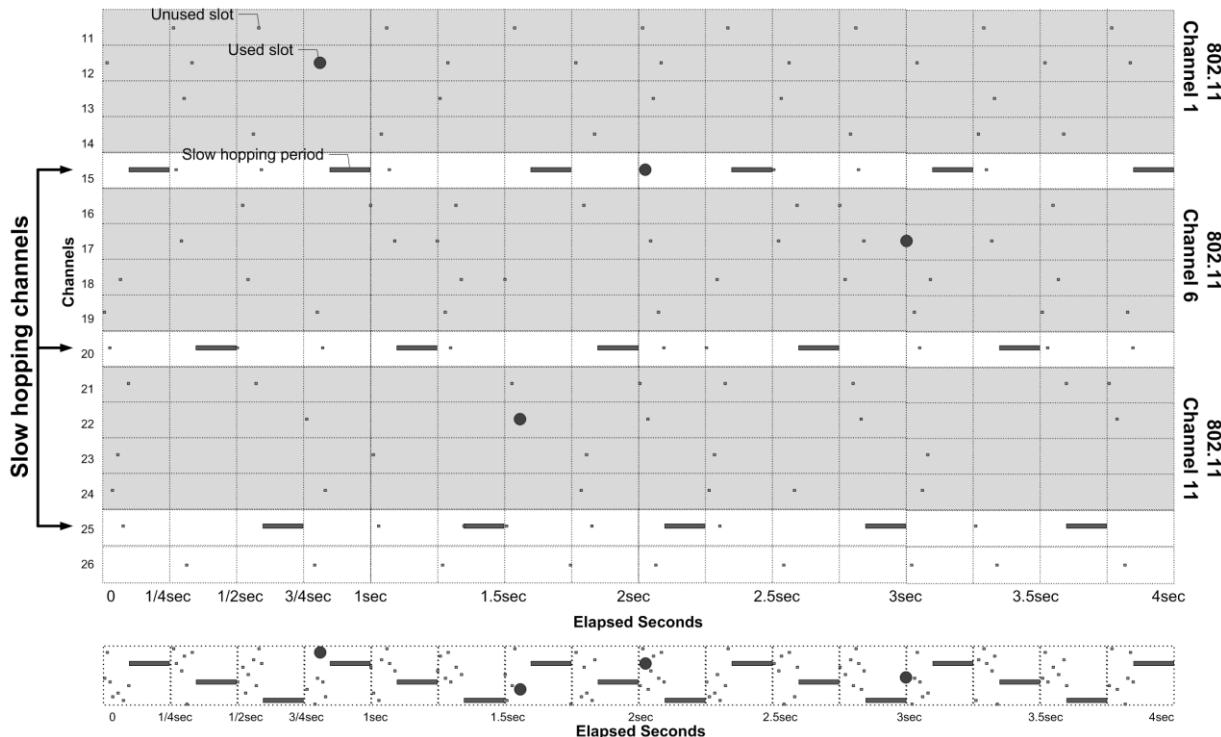
The bottom portion of Figure 63 illustrates that the hopping pattern can be used repeatedly as time progresses. Superframe length and timeslot usage by the DL is not necessarily tied to lower DL cyclical constructs such as hopping patterns or 250 ms alignment intervals. (See 9.1.9.1.3 for a discussion of alignment intervals.)

Figure 64 illustrates the use of slow hopping. Each channel is used over multiple timeslots.

**Figure 64 – Slow hopping**

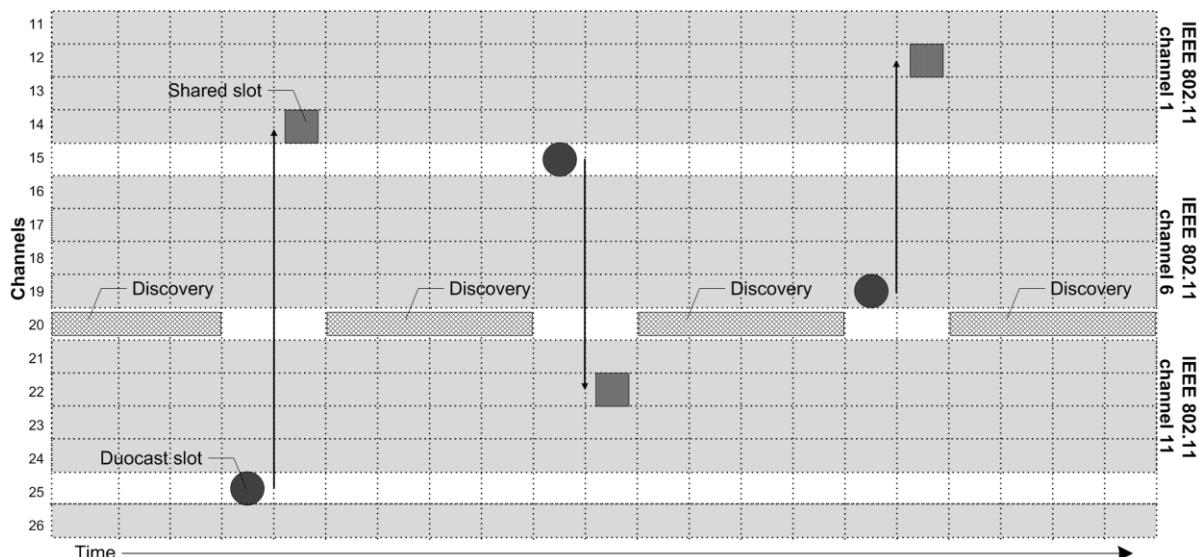
Slow-hopping periods can span a 250 ms alignment interval. (See 9.1.9.1.3 for a discussion of alignment intervals.) In such cases, slow-hopping periods are not interrupted by idle periods; that is, if a slow hopping period traverses the edge of an alignment interval, the radio does not turn off during the normal idle period.

Figure 65 illustrates a hybrid system, in which slotted hopping and slow hopping are combined. In this example, within each 250 ms alignment interval, a number of timeslots, each on a different channel, is followed by a slow hopping period on a single channel.



**Figure 65 – Hybrid mode with slotted and slow hopping**

The order in which slotted hopping and slow hopping are combined is flexible; that is, slow-hopping periods need not follow slotted hopping timeslots. Rather, the two may be used in any sensible combination. For example, Figure 66 shows an example configuration, where a device switches between slow hopping and slotted hopping.



**Figure 66 – Combining slotted hopping and slow hopping**

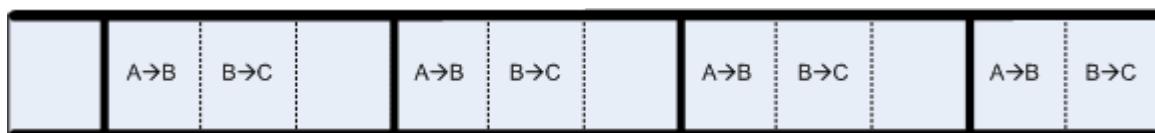
In the example of Figure 66, slotted hopping is used when broadcast, duocast, or contention-based communication timeslots are allocated explicitly. When timeslots are not allocated, the receiver listens on channel 20 to facilitate neighbor discovery. (See 9.1.9.4.7 for a discussion of duocast.)

## 9.1.8 Superframes

### 9.1.8.1 General

A superframe is a collection of timeslots repeating in time. The number of timeslots in each superframe cycle (its length) determines how quickly each superframe cycle repeats, thus setting a communication schedule for devices that use the superframe. For example, a superframe that cycles every 500 ms will allow each device that uses a single superframe timeslot to communicate at 500 ms intervals.

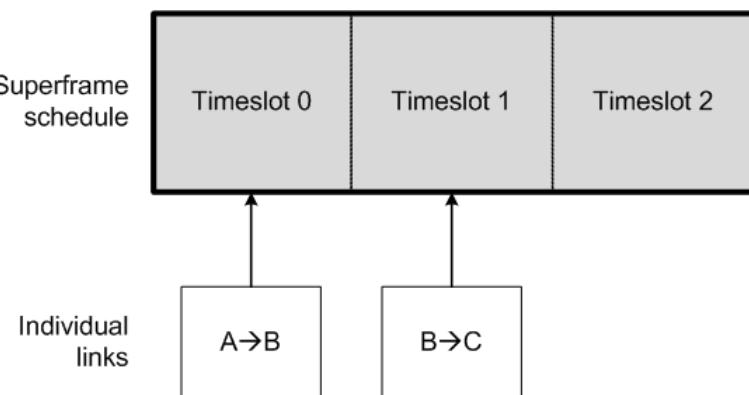
When a superframe is created, it is associated with a superframe ID for identification. Every new superframe instance in time is called a superframe cycle. Figure 67 shows how devices may communicate in an example three-timeslot superframe.



**Figure 67 – Example of a three-timeslot superframe**

In Figure 67, device A may communicate with device B during the first timeslot of each superframe cycle. Device B may communicate with device C during the second timeslot of each cycle, and the third timeslot of each cycle is idle (unassigned). Every three timeslots, the cycle repeats.

Figure 67 shows timing cycles and communication links within the same structure, which is a conceptual view. The DL actually represents the superframe cycles and communication links as separate but related configurable entities. Figure 68 illustrates this data structure, clarifying the distinction between superframes and links, for the same three-timeslot superframe as in Figure 67.



**Figure 68 – Superframes and links**

As shown in Figure 68, superframes refer to a collection of timeslots. Links refer to the use of superframe timeslots for communication between a specific pair of devices. A timeslot is a period of time. A superframe is a cyclic schedule of timeslots. A link describes a specific activity that repeats within a superframe's cyclic schedule.

The system manager configures matching sets of links among a collection of devices that communicate with each other. For example, a link on Device A may be configured to transmit DPDUs to Device B on a particular superframe cycle. On the same cycle, Device B should have a link that is configured to listen for an incoming DPDUs. These two links are matched in the sense that the system manager has configured these two related operations to occur in tandem.

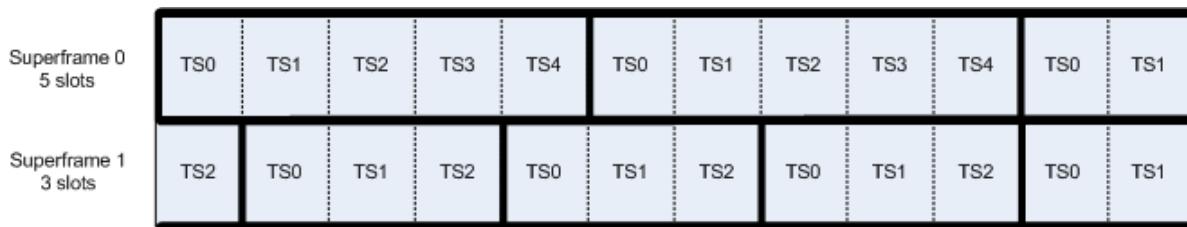
Several performance parameters are determined by superframe period and how links are assigned to superframes. In general, shorter-period superframes result in lower DPDU latency and increased digital bandwidth, at the expense of increased energy consumption and more concentrated allocation of digital bandwidth. Longer-period superframes generally result in higher latency and lower digital bandwidth, but with reduced energy consumption and less concentrated allocation of digital bandwidth. These tradeoffs should be carefully considered when determining superframe period and link density within a superframe.

A given device may contain several concurrent superframes of different lengths. A link with one timeslot within a superframe of length L slots repeats twice as frequently as a link with one timeslot within a superframe of length 2L slots, thus allowing for twice the throughput per link.

A device may participate in one or more superframes simultaneously, and not all devices in a DL subnet need to participate in all superframes. By configuring a device to participate in multiple overlapping superframes of different lengths, it is possible to establish different communication schedules that all operate simultaneously.

Superframes are numbered, but these superframe numbers are limited in scope to the device where the superframe is used. Since the scope of a superframe number is a single device, a neighboring device may use the same superframe number for a completely different purpose. Superframes can be added, removed, activated, and deactivated while the network is running.

Figure 69 shows how timeslots in different superframes are aligned, even though the superframes may cycle at independent rates.



**Figure 69 – Multiple superframes, with timeslots aligned**

NOTE Timeslot alignment is a result of defining all slots as same duration and aligning timeslots with TAI time every 250 ms (see 9.1.9.1.3). Superframes with timeslots of different durations may be used simultaneously within a network. However, the designers of this standard considered only configurations wherein one timeslot duration is used at a time in a given DL subnet. Devices are not required to support multiple timeslot durations simultaneously.

A device with multiple links in a timeslot may encounter link collisions when two or more links coincide. To address such situations, each link is assigned a priority. A higher-numbered link priority means the link takes precedence over a link with a lower-numbered priority. In the event of a link collision, the link with the higher-numbered priority is used. If two links have the same priority, a superframe priority is used. See 9.1.8.5.

In addition to link priority, each message is assigned a priority at its DL point of origin. Once a link is selected based on link priority, the DPDU priority is used to weigh the relative importance of DPDUs on the queue that can use the link.

### 9.1.8.2 Exponential backoff

Links may be shared or dedicated. On the receiver side, there is no structural difference between shared versus dedicated links. On the transmitter side, an exponential backoff bit in the link indicates whether the link is shared or dedicated. If a link is shared, as indicated by the link's exponential backoff bit of 1, the transmitter shall use exponential backoff for retries using that link. If a link is dedicated, as indicated by the link's exponential backoff bit of 0, the transmitter shall not use exponential backoff in that link.

It is possible, and sometimes reasonable, for a system manager to configure multiple devices to transmit at the same time and in the same radio channel, without setting an exponential backoff bit in the applicable links. In this subclause, such links are called dedicated merely to indicate that exponential backoff is not applied to those links.

Exponential backoff shall be applied when, and only when, a unicast DPDU is transmitted in a shared link and does not receive an ACK or a NACK, indicating the possibility of a collision. A unicast transmission that does not occur due to CCA shall be treated as equivalent to an unsuccessful transmission in the context of exponential backoff. Exponential backoff is intended to resolve such collisions. Exponential backoff shall operate on a per-neighbor basis, applied to all DPDUs in the message queue regardless of priority.

For each neighbor, the DL tracks a backoff exponent and a backoff counter, called `BackoffExponent[Neighbor]` and `BackoffCounter[Neighbor]` herein. `BackoffExponent` and `BackoffCounter` are not externally exposed, and therefore are not included in the DL object model.

`BackoffExponent[Neighbor]` and `BackoffCounter[Neighbor]` are set to zero every time an acknowledged unicast DPDU is successfully transmitted to a particular neighbor in a shared link.

BackoffCounter[Neighbor] of zero allows the device to send a DPDU at the next transmission opportunity in a shared link. Following an unsuccessful transmission to the neighbor in a shared link, the device increments BackoffExponent[Neighbor], and then sets BackoffCounter[Neighbor] by selecting a random number from 0 to  $2^{(\text{BackoffExponent}[Neighbor]-1)}$ . For each transmit opportunity in a shared link, BackoffCounter[Neighbor] is decremented until it reaches zero. If a transmit opportunity is in a dedicated link (no exponential backoff indicator), the device may use the link regardless of BackoffCounter[Neighbor] value. An attribute `dlmo.MaxBackoffExp` sets a ceiling on the value of BackoffExponent[Neighbor].

Retry behavior can be configured by the system manager. DL attributes that relate to retries include:

- `dlmo.MaxBackoffExp`. The maximum value for `BackoffExponent[Neighbor]`.
  - `dlmo.MaxLifetime` and `dlmo.Graph.MaxLifetime`: Maximum lifetime of a DPDU. A DPDU that is being forwarded shall be deleted if held in the DL message queue for longer than `MaxLifetime`. `dlmo.MaxLifetime` provides a default value for the DL. A non-null value for `dlmo.Graph[].MaxLifetime` indicates that `dlmo.MaxLifetime` shall be overridden for DPUS following a particular graph.

Operation of exponential backoff is illustrated in the following pseudocode:

```

// For each neighbor, independently
BExp[Nei] = 0; // BackoffExponent[Neighbor] in text
BCnt[Nei] = 0; // BackoffCounter[Neighbor] in text
For each timeslot (
If (transmit link and DPDUs match) // See 9.1.8.5
    If (not exponential backoff link) (
        // Dedicated link
        Attempt to transmit DPDUs using link;
        If (transmit was successful) remove DPDUs from queue;
    )
    Else (
        // Shared link
        If (BCnt[Nei] > 0) BCnt[Nei]--;
        Else (
            Attempt to transmit DPDUs in link;
            If (transmit was successful) (
                Remove DPDUs from message queue;
                BExp[Nei]=0;
                BCnt[Nei]=0;
            )
            Else (
                // Transmit failed; exponential backoff
                If (BExp[Nei] < MaxBackoffExp) BExp[Nei]++;
                BCnt[Nei] = Random (0, 2^(BExp[Nei]-1));
            )
        )
    )
)
Delete all messages beyond MaxLifetime;
If (no queued DPDUs for neighbor) (BCnt[Nei]=0; BExp[Nei]=0);

```

NOTE As described in 9.1.8.5, a link can be configured as a Transmit/Receive (T/R) link, which is a compressed representation of a transmit link and a receive link. Logically, T/R links are processed as two independent links.

### 9.1.8.3 Superframe channel use

Timeslots within a superframe are associated with a slow or slotted hopping pattern, as well as an offset into that pattern.

From the perspective of each device using a superframe, there is a baseline hopping pattern offset, which may vary from device to device and which may be overridden with an alternative offset applied to a link or collection of links within a superframe.

A given unicast transaction occurs on a single channel, with the DPDU and acknowledgement(s) both transmitted on the same channel.

A superframe is not limited to one channel at a time; rather, a superframe is a two-dimensional structure indicating time and channel. This was previously illustrated in Figure 62 (see 9.1.7.2.5).

Figure 62 shows a superframe, with time on the horizontal axis and channel on the vertical axis. The superframe encompasses all of the channels over the length of that superframe. As shown in Figure 62, 16 devices may use 16 different offsets from hopping pattern A; the superframe encompasses all of the channel assignments for all of the superframe timeslots.

The default channel offset may be different for transmitting versus receiving and may vary by link.

The hopping pattern is not necessarily related to the length of the superframe. Referring to Figure 62, a superframe might be configured as 25 timeslots long, even though hopping pattern A is only 16 hops long.

For frequency diversity, superframe length and hopping pattern length may be configured to be relatively prime, that is, with no common factors. As a counter-example, consider a configuration wherein superframe length is 25 timeslots, with a hopping pattern repeating on a 15 channel cycle, resulting in a superframe schedule where only 3 of the 15 available channels are ever used. Such an arrangement can cause regulatory issues in situations where use of all channels is required by each device.

### 9.1.8.4 Organizing superframes

#### 9.1.8.4.1 General

Two general superframe types are supported:

- Slotted hopping, which makes optimal use of available digital bandwidth and supports battery-powered routers.
- Slow hopping, intended for routers with available energy to run their receivers continuously during a given period. Slow hopping allows neighboring devices to operate with less exacting time synchronization requirements, particularly during the neighbor discovery process.

Hybrid configurations may be arranged by combining superframes, for example, one slotted and one slow. Slotted and slow hopping will be discussed separately, followed by some examples of hybrid configurations.

NOTE In the marketplace, slow hopping is sometimes referred to as CSMA, and slotted hopping as TDMA. These terms are not used in this standard, except to the extent that CSMA-CA is supported by the standard in a literal sense. (See 9.1.9.4.8.) Slow hopping is built on a TDMA base, slotted hopping includes CSMA aspects. The solution designer is free to mix the approaches.

#### 9.1.8.4.2 Superframe scope

Superframes are commonly discussed as abstractions that span several devices. Nonetheless, while the superframe may be conceptualized at the DL subnet level, the scope of the superframe data structure is limited to each device. A superframe is instantiated as a

data structure on a single device that independently drives its DL state machine. A device's superframe definitions need to relate to those of its neighbors so that devices communicate at the same time. Superframe definitions within each device are numbered, but that numbering is only needed by the DMAP for table read/write operations and by other objects and attributes within the device that refer to the superframe.

A superframe may be contrasted with a routing graph's scope. A graph ID, unlike a superframe ID, is carried in a DPDUs DROUT header, and a graph ID shall be consistent and unique in all devices that use the graph. No such constraints apply to a superframe.

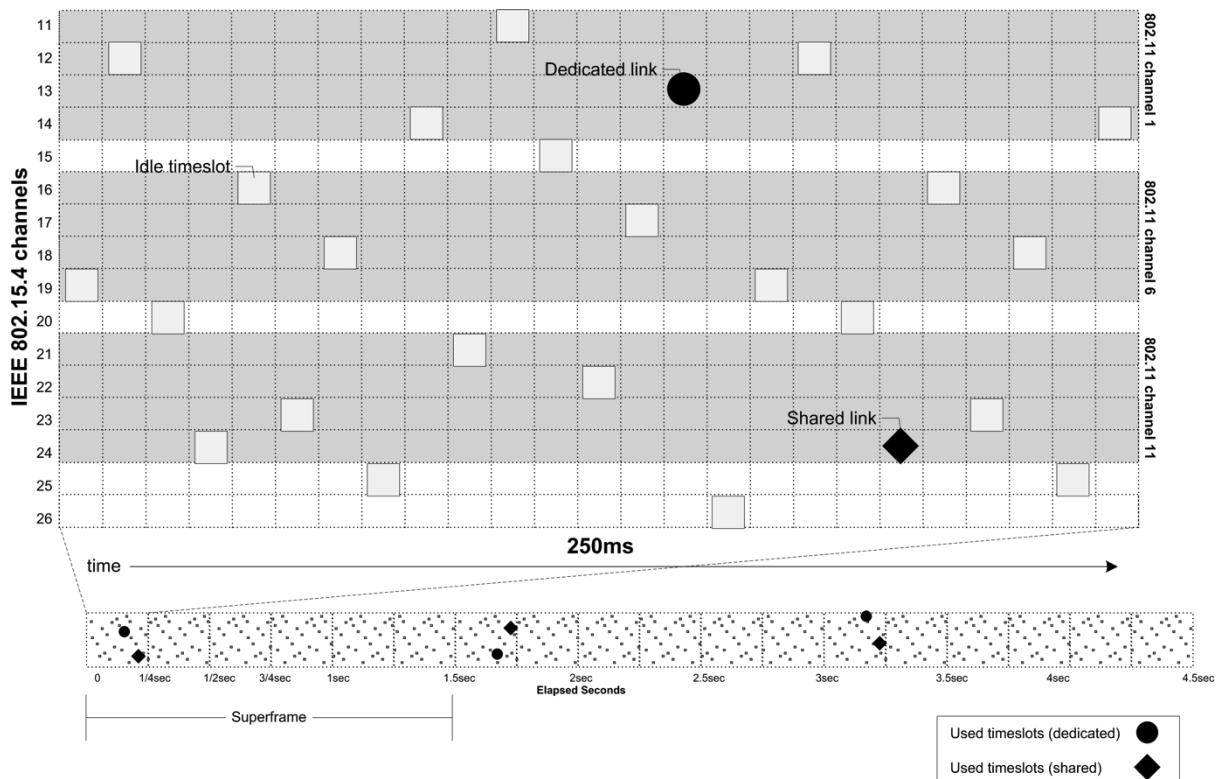
#### 9.1.8.4.3 Blocks of contention-based capacity

Mains powered routers may sensibly operate their receivers continuously. As its lowest priority superframe, such a router may support a superframe comprised partially or entirely of receive links. The router's neighbors may maintain corresponding shared transmit links to the router. Such a configuration results in blocks of contention-based digital bandwidth available to the routers neighbors. Slow hopping or slotted hopping may be used in a superframe of that type. Channel hopping offset may be selected to avoid collisions between dedicated links versus a general inventory of shared links.

#### 9.1.8.4.4 Slotted hopping

Slotted hopping uses channel-hopping superframe timeslots of equal duration. Each superframe timeslot uses a different radio channel in a hopping pattern. In slotted hopping, each superframe timeslot is intended to accommodate one transaction, including a DPDUs and its acknowledgement(s).

Figure 70 illustrates a slotted hopping superframe from the perspective of one device, which may be a router.



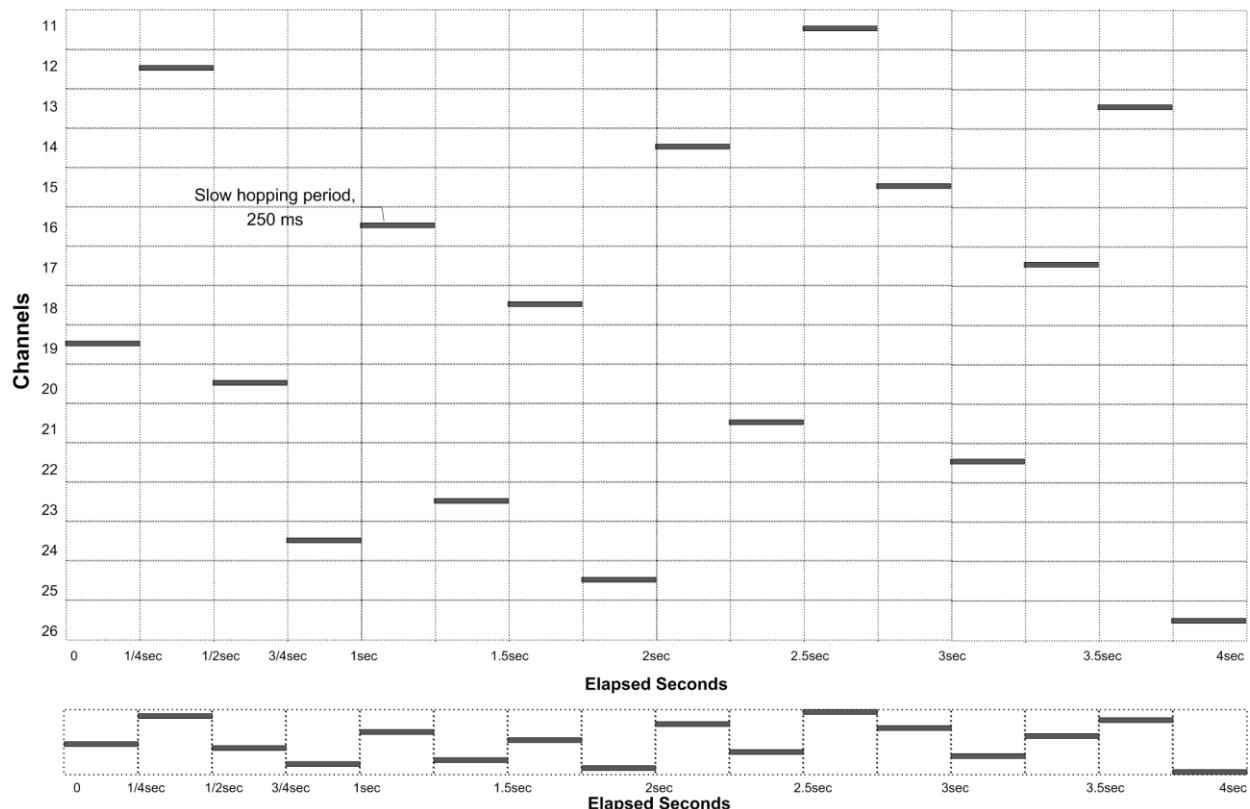
**Figure 70 – Slotted hopping**

In this example, the superframe is 1.5 s long. Timeslots with link assignments are depicted with circles (dedicated links) and diamonds (shared links). As Figure 70 shows, from the perspective of a single field device, many superframe timeslots might be left idle. Timeslots with link assignments repeat at a fixed interval defined by the superframe length.

#### 9.1.8.4.5 Slow hopping

In slow hopping, a collection of contiguous superframe timeslots is grouped on a single radio channel. Each such collection of superframe timeslots is treated as a single slow hopping period.

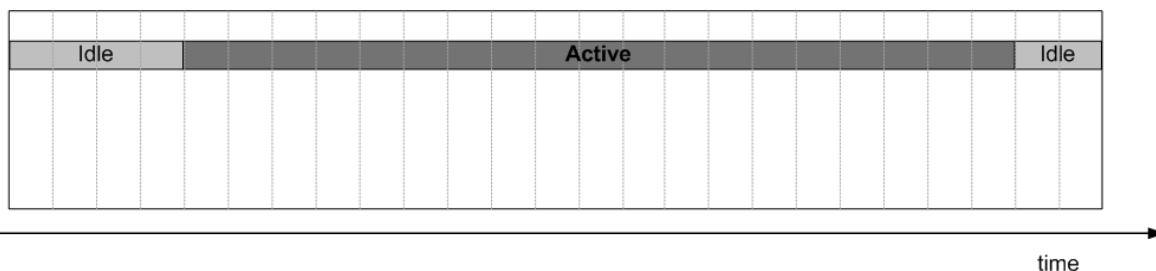
Figure 71 illustrates slow hopping.



**Figure 71 – Slow hopping**

Timeslots in a slow hopping superframe are generally shared, providing immediate, contention-based channel bandwidth on demand to a router's immediate neighbors.

Figure 72 shows the main components of a slow hopping superframe.

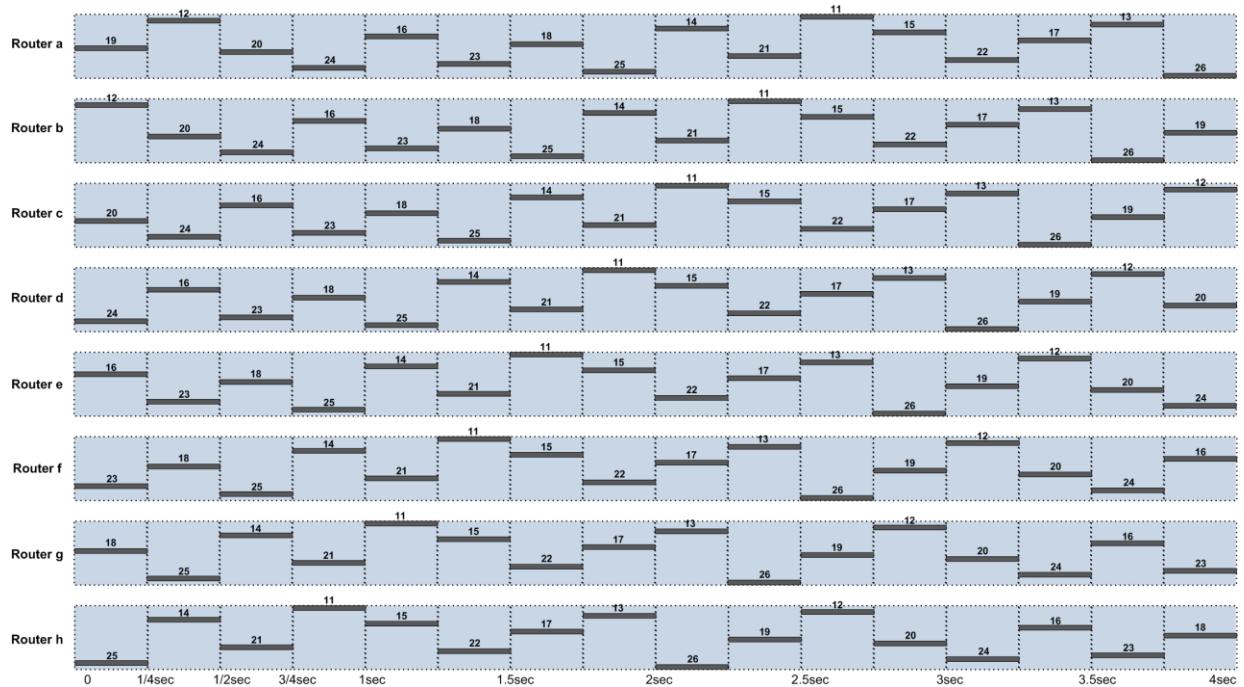


**Figure 72 – Components of a slow hop**

A baseline slow hopping period has a fixed length, with a fixed number of idle timeslots (which may be zero) at the beginning and end of the hop. The example of a slow hopping period shown in Figure 72 is comprised of 25 timeslots, including 3 idle timeslots at the beginning, 20 active timeslots in the middle, and 2 idle timeslots at the end. The idle timeslots are intended to support hybrid configurations where slow hopping superframes are paired with slotted hopping superframes, with the slotted hopping timeslots scheduled for use during the idle periods of the slow hopping superframe.

NOTE Idle periods as described here can be configured by matching superframe and channel hopping phases, and defining links that match the desired active range.

Figure 73 illustrates how a collection of up to 16 slow hopping routers may be configured to avoid collisions.



**Figure 73 – Avoiding collisions among routers**

Each router may use a different offset into the hopping pattern. With a 16-channel hopping pattern, each of up to 16 routers may be configured with a different offset into the pattern, so that no two routers use the same channel at the same time.

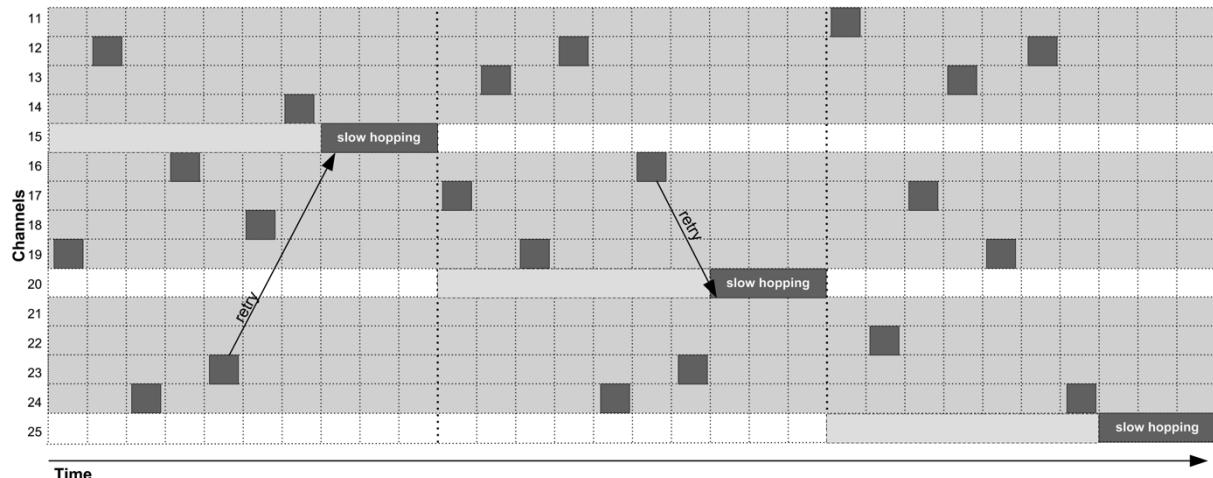
See 9.4.3.5.4 for more detail on slow hopping.

#### 9.1.8.4.6 Hybrid hopping configurations

Hybrid configurations may use combinations of the slotted hopping and slow hopping superframes.

Hybrid configurations are typically arranged so that slotted hopping links are allocated for scheduled, periodic messaging. This may leave blocks of lightly-used slow hopping capacity, available on a contention basis, for less predictable uses such as alarms and retries.

The example in Figure 74 illustrates how slotted hopping and slow hopping superframes may be combined.



**Figure 74 – Hybrid configuration**

In Figure 74, slotted hopping has been overlaid on a slow hopping background. The slow-hopping periods fill in the time between periodic collections of dedicated superframe timeslots.

In this configuration, if an attempted transmission in a dedicated timeslot fails, the succeeding slow hopping period can be used to retry the transmission. In Figure 74, two of the superframe timeslots are shown with retries on different channels during the subsequent slow hopping period.

#### 9.1.8.4.7 Superframes and spectrum management

The term spectrum management refers to the ability of the system manager to configure a DL subnet to block unwanted channels from operation in a DL subnet.

Each superframe includes a channel map, called Superframe[].ChMap, that is a bit mask of channels that shall be included and excluded in the hop sequence that is referenced by the superframe. Excluded channels have the effect of shortening the hop sequence.

For example, a superframe channel map that includes channels 11-25 and excludes channel 26 has the effect of shortening the hop sequence by removing channel 26 from the hop sequence. More generally, the system manager may eliminate any collection of channels from the hop sequences that are referenced by superframes in a DL subnet, with the result of removing those channels from operation.

There is also a channel map in the DeviceCapability attribute that is reported to the system manager when the device joins the network. The DeviceCapability channel map does not shorten any hopping sequence used by the DL, but rather is a signal to the system manager that, for regulatory reasons, any link using one of the excluded channels will be treated as idle.

The system manager may also block use of certain channels through the attribute dlmo.IdleChannels. Unlike the channel map in the superframes, dlmo.IdleChannels does not cause hop sequences to be shortened; rather, it causes links on designated channels to be treated as idle. dlmo.IdleChannels is intended to provide a quick way for the system manager to disable certain channels in a way that does not require subnet-wide coordination of revised hop sequences.

Channel-specific diagnostics, as described in 9.4.2.27, provide the system manager with information to support spectrum management.

#### 9.1.8.5 Data link layer message queue operation

DL routers compliant with this standard shall support a DL message queue. This message queue has some attributes that can be configured by the system manager, affecting how the queue operates.

The standard does not generally specify internal device mechanisms. However, to a limited degree, the DL message queue is specified by the standard. The system manager can configure the DL message queue to achieve particular quality of service objectives, and a limited model of message queue behavior is implicit in the configuration alternatives provided to the system manager.

When a field device receives a unicast DPDU from its neighbor, it first assesses whether the DSDU should be passed to the network layer or forwarded to another device through the DL, as described in 9.3.3.6.

If the DPDU needs to be forwarded, the device then evaluates whether the DPDU should be accepted or NACKed, in part based on the available capacity of the DL message queue. DPDUs shall be NACKed if the DL message queue has run out of capacity for DPDUs of that type.

The DL reports its forwarding queue capacity to the system manager when the device joins the network, through the attribute `dlm0.DeviceCapability`, field `QueueCapacity`. This externally reported queue capacity does not include portions of the queue that are reserved for the device's internal use. For example, the DL message queue in a particular device might report that it has a queue capacity for five DPDUs. The system manager is then able to configure those five positions in the queue. This nominal capacity refers only to a portion of the message queue that is exclusively used to route DPDUs through the device. In practice, the actual device has additional message queue capacity space that it does not report to the system manager, because the device also needs to handle DPDUs on its own behalf. Unreported message buffer capacity, for the device's own use, is considered an internal device matter and is not allocated by the system manager. The system manager assumes that the device has sufficient message queue capacity for its own use when contracts are granted.

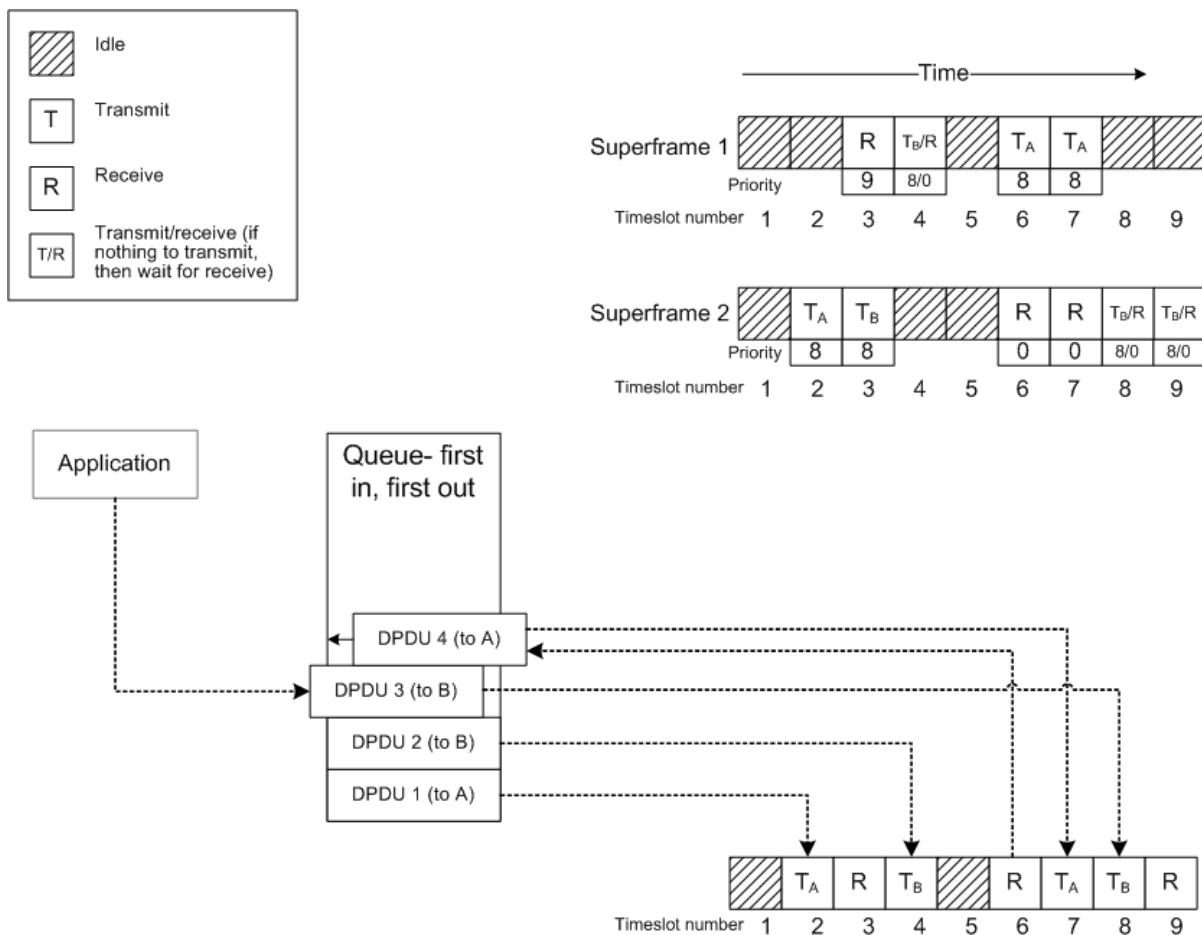
Consider the example of a field router with a reported DL message buffer capacity of eight DPDUs. The actual buffer capacity may be twelve DPDUs, in which case the difference between reported and actual capacity (four DPDUs) is for the device's own use. In this example, the system manager might reasonably configure the device's nominal buffer capacity as follows:

- No more than 3 of the 8 buffers will be used to forward DPDUs with priority  $\leq 2$ .
- No more than 5 of the 8 buffers may be used to forward DPDUs with priority  $\leq 5$ .

See 9.4.2.26 for further discussion of queue buffer capacity and priority levels.

In addition, for a finer grained degree of control, the system manager can designate a certain number buffers exclusively to forward DPDUs that are being routed along specific graph, as described in 9.4.3.7.

Figure 75 provides an overview of how links interact with an implicit DL message queue within a device.



**Figure 75 – Timeslot allocation and the message queue**

As shown in Figure 75, DPDUs are held in a message queue until transmit links become available. DPDUs are placed in the queue in the order they are received. Generally, retrieval of DPDUs from the queue is first-in, first-out (FIFO).

NOTE 1 This simplified example shows only a single destination address for each DPDUs. In practice, each DPDUs on the message queue is actually a candidate for links to multiple neighbors.

Consider the example of DPDUs 3. DPDUs 3 originates in an application and enters the DL through the DDSAP (Figure 53). When the DPDUs is processed by the DL, it is placed in the DLs message queue. Based on the DPDUs ultimate destination, the DL determines that the DPDUs needs a link to device B for its next hop. DPDUs 1 and 2 are already on the queue, so DPDUs 3 is queued behind them. When a link to device B becomes available, DPDUs 2 will be sent before DPDUs 3.

Each DPDUs on the queue is assigned a priority by the originating device. This simplified tutorial assumes that all DPDUs have equal priority. In actual practice, link priority takes precedence over DPDUs priority, in the sense that DPDUs are not considered for transmission until after a transmit link is selected. Once a transmit link has been selected, the DPDUs on the queue is selected based first on priority, and then on a FIFO basis if multiple candidate DPDUs have the same priority.

DPDUs 4 shows an example where a DPDUs is received in one timeslot and is available to be forwarded in the next timeslot. In general, a DPDUs received in one timeslot (timeslot N) shall be available on the queue for forwarding in next timeslot (timeslot N+1). An exception is allowed when default timeslot template 3 is used (see Figure 157). In that case, the receive transaction might be incomplete at the start of the next timeslot. A DPDUs received starting in one timeslot (timeslot N), using default timeslot template 3, shall be available on the queue for forwarding in the timeslot flowing the next timeslot (timeslot N+2).

In Figure 75, two superframes are shown, and each superframe is associated with a series of links. For example, timeslot 3 in superframe 1 is associated with a receive link (R), while timeslot 3 in superframe 2 is associated with a transmit link to device B ( $T_B$ ). Idle slots do not have defined links.

Each link has a priority. The attributes `dlmo.LinkPriorityXmit` and `dlmo.LinkPriorityRcv` provide default link priorities, which may be overridden for any particular link. The example in Figure 75 mostly shows the default priorities of 0 for receive links and 8 for transmit links.

The system manager should typically assign lower priority to receive links (R) than to transmit links (T), thus giving precedence to servicing outgoing DPDUs on its queue. This is not strictly required however. For example, if a latency-critical incoming flow is scheduled for a particular timeslot, the system manager may configure receive links with higher priority in that case. As an illustration in Figure 75, the third link in Superframe 1 is assigned a priority of 9, giving this particular receive link a higher priority than a transmit link at the same time.

Transmit/receive (T/R) timeslots use a compressed format to combine a transmit link and a receive link. Logically, a T/R link is two links, with the receive part of the link having a priority of `dlmo.LinkPriorityRcv` which defaults to zero. For example, if a timeslot has a high priority  $T_A/R$  link and lower priority  $T_B$  link, the  $T_B$  link has higher priority than the R part of the  $T_A/R$  link.

**NOTE** Baseline operation is that a transmit D\_pdu and a link are matched at the start of a timeslot, and the timeslot is considered to be a transaction that is run to completion according to the link configuration. In the event that a transmit link is aborted due to CCA, an optimized implementation may complete the timeslot using a receive link that is valid for the same timeslot.

Once a D\_pdu has been transmitted and acknowledged, the transaction is deemed successful and the D\_pdu is deleted from the queue. This example assumes that all transactions are successful.

The box at the bottom right of Figure 75 illustrates how links are used in this example.

- Timeslot 1: The first link in both superframes 1 and 2 are idle; therefore, the first timeslot is idle.
- Timeslot 2: The second link in superframe 1 is idle, but there is a transmit link for device A ( $T_A$ ) in superframe 2. There is also a D\_pdu to device A in the queue; therefore, the second timeslot is assigned for transmission to device A, and D\_pdu 1 is sent.
- Timeslot 3: Superframe 1 has a receive link, and superframe 2 has a transmit link. The link in superframe 1 takes precedence due to its priority, so timeslot 3 is used to listen for incoming DPDUs. (As described above, receive links are usually assigned lower priority than transmit links. This example illustrates how a system manager can give priority to a receive link, for example, to service an incoming flow.)
- Timeslot 4: Superframe 1 has a  $T_B/R$  link, indicating that it should transmit if there is a D\_pdu for device B in the queue. D\_pdu 2 is sent.
- Timeslot 5: Both superframes are idle in timeslot 5, so the timeslot is idle.
- Timeslot 6: Superframe 1 designates a transmission link to device A, but there is no longer a D\_pdu for device A in the queue. Since there is nothing useful for the transmit link to do in this slot, the link in Superframe 2 is used. The device receives an inbound D\_pdu, determines that its next hop is to device A, and places the D\_pdu on the queue.
- Timeslot 7: Now that there is a D\_pdu for device A on the queue, the transmit link in superframe 1 gets priority and D\_pdu 4 is sent. Note that D\_pdu 3 was skipped over because no  $T_B$  link has become available yet.
- Timeslot 8: Now that a  $T_B$  link is available, D\_pdu 3 is sent.
- Timeslot 9: The  $T_B/R$  link results in a receive slot, because there is no D\_pdu to device B on the queue.

This example was simplified in some essential respects.

- As noted above, DPDUs priorities were assumed to be equal. In practice, if two DPDUs on a message queue both match a given link, the DPDUs with the higher priority is transmitted. The FIFO queue position is relevant only for DPDUs of equal priority.
- If a unicast DPDUs does not receive an acknowledgement, it stays on the queue and the DL retry strategy is applied. See 9.1.8.2.
- A link may be configured for use by a particular graph. In that case, DPDUs with that graph ID shall be granted prioritized or exclusive access to the link. See 9.4.3.7.
- A field device may be designed to skip links on radio channels with a history of subpar connectivity. A field device may also skip links in order to retry on an alternative link. See 9.1.7.2.4.

Operation of queue processing is illustrated in the following pseudocode:

```

For each timeslot (
    Order DPDUs on queue by priority;
    // FIFO within priority

    Order links by priority;
    // Treat T/R link as two links, with receive side
    // of link assigned link priority dlmo.LinkPriorityRcv;
    // Within link priority, order by superframe priority,
    // then superframe number (highest first);

    For each link, in priority order
        If it is a receive link (
            Use the receive link;
            Done with timeslot;
        )
        Else // it is a transmit link
            For each DPDUs, in priority order
                If link matches DPDUs (
                    Use the link;
                    Done with timeslot;
                )
)
)

```

## **9.1.9 Data link layer time keeping**

### **9.1.9.1 Timing**

#### **9.1.9.1.1 General**

The DL propagates and uses International Atomic Time (TAI) for its internal operation, and also provides TAI time as a service (through the DMAP) to wireless field devices compliant with this standard.

Devices within a DL subnet may be configured to track a shared sense of time to within a few milliseconds of each other, to support sequence of event reporting and other application-layer coordinated operations within the scope of a subnet. Synchronized time among immediate neighbors is also essential for operation of the wireless protocol.

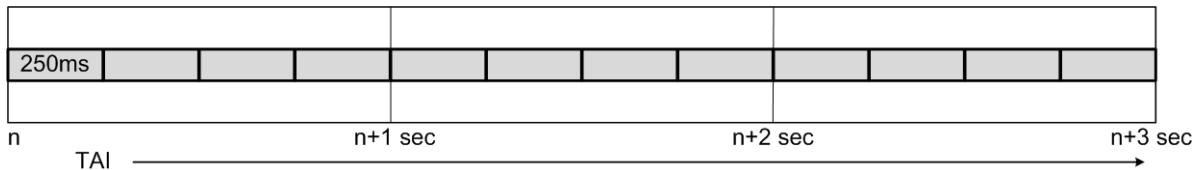
Slotted hopping requires tight time synchronization among immediate neighbors. However, the internal clock of a non-routing field device that has been disconnected from DL subnet operation for more than a few minutes may have drifted by tens or hundreds of milliseconds or more in relation to the overall DL subnet clock. Slow hopping or hybrid hopping configurations support the continued operation of such devices.

#### **9.1.9.1.2 International atomic time**

In this standard, time is based on International Atomic Time (TAI) as the time reference. See 5.6.

#### **9.1.9.1.3 Alignment intervals**

In this standard, one-second TAI increments are divided into 250 ms (1/4 s) alignment intervals, wherein the 250 ms (2<sup>-2</sup> s) cycles shall align with nominal TAI s as shown in Figure 76.



**Figure 76 – 250 ms alignment intervals**

Continuous control loops in the process industries that have been ISA's traditional focus are frequently based on fixed-period computation of control outputs. These process loops repeat at rates of 4 Hz (250 ms), 1 Hz (1 s) or multiples of either 4 s or 5 s. Process monitoring is typically mapped onto this same 4 Hz structure.

NOTE Applications requiring slightly higher loop rates, such as compressor surge control loops running at 12 Hz, can be supported by scheduling multiple communications opportunities per 250 ms cycle.

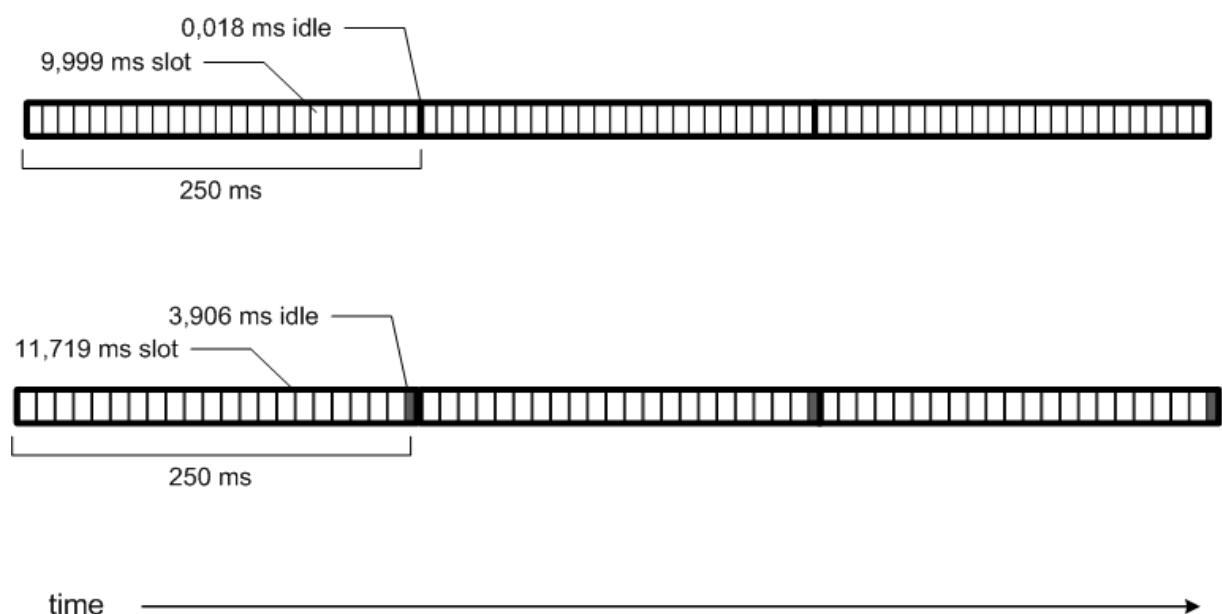
#### 9.1.9.1.4 Timeslot duration, timeslot alignment, and idle periods

Within 250 ms alignment intervals, time is divided into timeslots of configurable but equal duration. A timeslot is a time interval of predefined duration used to send or receive a DPDU and any corresponding acknowledgement(s). Timeslots are generally shared by at least one pair of devices that communicate during the allocated time. The DL organizes these timeslots into superframes, which are collections of timeslots with a common period and potentially other common attributes.

Timeslot durations are configured during network setup. Normally, during a given period of operation, all timeslots within a DL subnet have the same duration. Timeslot duration is configured in units of  $2^{-20}$  s ( $\sim 0.95 \mu\text{s}$ ).

NOTE 1 The DL binds timeslot duration to superframes, and nothing in the standard prevents multiple superframes with different timeslot durations from being active simultaneously within a DL subnet. However, this standard considers only configurations wherein a single timeslot duration is used in a given DL subnet. A device supporting multiple timeslot durations simultaneously, such as a backbone router or a bridge between two subnets, may be modeled as containing multiple DLs running in parallel.

Timeslots align with the 250 ms alignment intervals, but timeslot durations do not necessarily divide evenly into 250 ms. Therefore, the system inserts a short idle period every 250 ms as needed, thus realigning the timeslots to a 4 Hz cycle. This is shown in Figure 77 with two illustrative examples using different timeslot durations of 9,999 ms and 11,719 ms, respectively.



**Figure 77 – Timeslot durations and timing**

NOTE 2 Calculations for the idle times in Figure 77:

- (a)  $10485*25*2^{-20}$  s = 249.982 ms. Subtract from 250 ms to get 0.018 ms.
- (b)  $12288*21*2^{-20}$  s = 246.094 ms. Subtract from 250 ms to get 3.906 ms.

NOTE 3 A timeslot intended closely to approximate 10 ms should be 10485 units, or 9,999275 ms, in duration.

The idle period is not used by the system for scheduled operations; it is simply a short period inserted to ensure that timeslots align and repeat at 250 ms intervals. This makes it straightforward for the system manager to organize collections of timeslots that repeat at exact multiples of 250 ms.

Slow-hopping periods can span an alignment interval. In such cases, slow-hopping periods are not interrupted by idle periods; that is, if a slow hopping period traverses the edge of an alignment interval, the receiver's radio continues operation through the idle period.

#### 9.1.9.1.5 Scheduled timeslot time

Each DL timeslot has a scheduled fractional-second start time, which is called the scheduled timeslot time. Both the DL and higher-layer protocols use this coordinated time sense as security material, thus providing replay protection. See 7.3.2.6.

Since the 250 ms intervals align with TAI quarter-seconds, scheduled timeslot timing can be derived from the timeslot duration. For example, if timeslot duration is 9,999 ms, scheduled timeslot times in the second quarter-second of TAI time  $t$  is  $t,250000$  s,  $t,259999$  s,  $t,269998$  s, etc.

NOTE This mixed-radix time representation facilitates inter-conversion at higher layers where PDUs may span subnets with different data rates, different timeslot durations and/or different DL protocols.

The scheduled timeslot start time is in units of  $2^{-20}$  s (microsecond precision). Devices commonly use internal clocks based on a  $2^{15}$  Hz (32 kHz) crystal. Implementations may, and commonly will, round the scheduled timeslot start time to the nearest 32 kHz clock tick. This rounding is permitted on a per-timeslot basis, without adjusting the underlying timeslot schedule based on units of  $2^{-20}$  s. However, this rounding to 32 kHz is not permitted to accumulate over a 250 ms period, as shown in Table 102. This consideration is included within the  $\pm 96 \mu\text{s}$  jitter allowed by the standard; see 9.4.3.3.1.

**Table 102 – Approximating nominal timing with 32 kHz clock (INFORMATIVE)**

Nominal Timeslot offset (ms)	$2^{-20}$ s clock counts	$2^{-15}$ s clock counts Rounded	Actual Timeslot offset (ms)
0	0	0	0
10	10485*1	328	10.010
20	10485*2	655	19.989
30	10485*3	983	29.999
...	...	...	...
200	10485*20	6553	199.982
210	10485*21	6881	209.991
220	10485*22	7208	219.971
230	10485*23	7536	229.980
240	10485*24	7864	239.990

#### 9.1.9.2 Data link layer time propagation

##### 9.1.9.2.1 General

The DL uses TAI time for its internal operation, and provides a notion of TAI time as a service (through the DMAP) to wireless neighbors compliant with this standard.

In a DL subnet, devices may take on three functions in the time propagation process:

- DL clock recipient, a receiver of periodic clock updates through the DL; or
- DL clock source, a provider of periodic clock updates to DL neighbors; or

- DL clock repeater, a DL clock recipient that also acts as a DL clock source to some of its neighbors.

Additional information about time propagation can be found in 6.3.10.

### 9.1.9.2.2 Data link layer clock stability

The clock stability of the DL is the nominal clock stability of a wireless device, in the presence of periodic time corrections from the network.

The DL, when configured as a clock recipient, relies on the DL subnet to provide periodic time updates wirelessly. It may also use these time updates in calibration of its internal clock to account for conditions such as temperature, aging, and voltage.

The time reference for a subnet originates from one or more clock masters, which provide time that is monotonically increasing at a rate that closely tracks real time.

When the device joins the network, the DL reports its clock stability to the system manager as `dlmo.DeviceCapability.ClockStability` (called `ClockStability` here), which is in units of parts per million (ppm). `ClockStability` applies to any arbitrary 30 s period during the expected life of the device, under all conditions that a user might reasonably expect from the device's published specifications.

For example, if `ClockStability` reports that a device has a 10 ppm clock, then the device's clock shall be stable to within  $\pm 300 \mu\text{s}$  during any arbitrarily selected 30 s period.

`ClockStability` is reported as an envelope. For example, if `ClockStability` reports that a device has a 10 ppm clock, then the clock shall be stable to within  $\pm 300 \mu\text{s}$  at all instants during any arbitrarily selected 30 s period. This is intended to allow for devices that are subjected to occasional environmental shocks that can cause small clock discontinuities. Small discontinuities are acceptable, as long as they do not add up to more than `ClockStability`\*30  $\mu\text{s}$  at any time over any 30 s period.

NOTE 1 `ClockStability`, specified in units of ppm, can be converted to units of  $\mu\text{s}/\text{s}$ .

NOTE 2 Neighboring devices might have clocks that drift in opposite directions. Therefore, in the worst case, `ClockStability` is additive between pairs of neighboring devices. In practice clock repeaters are corrected periodically, which lessens that effect.

The standard assumes that clock drift is negligible within a timeslot.

`ClockStability` is reported to the system manager without caveats. If the device is specified to work under environmental stress, such as extremes of temperature or mechanical shock, then `ClockStability` shall reflect performance under such stress.

The attribute `dlmo.ClockExpire`, configured by the system manager, provides the maximum number of seconds that the DL can safely operate in the absence of a clock update. Normally, the system manager arranges that a device will maintain clock synchronization as a by-product of normal communication. However, when the DL fails to receive a clock update for an extended period of time, defined by `ClockExpire`, the DL should actively interrogate a DL clock source for a time update. Failure to do so will eventually result in loss of time synchronization with the DL subnet. `ClockExpire` defaults to a value that is appropriate for use during the join process.

A clock repeater should not send clock corrections to its neighbors through acknowledgements if it has not itself received a clock correction for a period that exceeds the `ClockExpire` attribute. If a clock repeater's clock has expired and it is polled for a time update, it should respond with a NACK1.

A DL with an expired clock should not be used as a clock repeater, but it may continue to operate in the subnet, albeit with a potential risk of losing synchronization with its neighbors. The attribute `dlmo.DLTimeout` provides the maximum amount of time that a device may reasonably continue operating in a subnet in the absence of a clock update. If the DL has not received a clock update for a period of time that exceeds `DLTimeout`, the device may reasonably reset itself to the provisioned state and initiate a search for a new network.

**NOTE 3** A device operating in a slow-hopping configuration may be configured to retain a network connection for extended periods of time, even with a clock that has drifted across timeslots.

### 9.1.9.2.3 Preferred and secondary clock sources

The system manager configures each DL clock recipient to treat one or several of its neighbors as DL clock sources. Such DL clock sources may be designated as preferred or secondary, based on the attribute `dlmo.Neighbor[].ClockSource`. Multiple neighbors may be designated as preferred DL clock sources. A DL clock recipient should adjust its clock value whenever it has an interaction with a preferred DL clock source.

The attribute `dlmo.ClockStale` defines a period of time that shall pass before a DL begins accepting clock updates from secondary DL clock sources. If, after a period of `ClockStale`, no clock update is received from any preferred source, the DL clock recipient should accept clock updates in its interactions with neighbors that are designated as secondary DL clock sources. Once a DL clock recipient accepts a clock update from a secondary DL clock source, it should continue to use that secondary DL clock source until either (a) it receives a clock update from a preferred DL clock source, or (b) the secondary DL clock source times out.

The attribute `dlmo.ClockStale` determines the timeout interval. For example, if `ClockStale` is set to 180 quarter-seconds by the system manager, then a DL clock source should not accept clock updates from a secondary DL clock source until it has not received a clock update from any preferred DL clock source for at least 180 quarter-seconds.

Each DL clock recipient can be configured to retain and periodically report statistics (see 9.4.3.9 for any of its preferred DL clock sources, including:

- A count of clock timeout events.
- A running average of clock corrections, a signed integer in units of  $2^{-20}$  s, indicating a bias if nonzero.
- The standard deviation of clock corrections, estimated in units of  $2^{-20}$  s, for example, a value that roughly accounts for approximately 68% of clock corrections.
- A count of clock corrections in excess of three such standard deviations.

### 9.1.9.2.4 Shared time sense during data link layer subnet operation

DL subnet transactions nominally occur in a DL timeslot, on a schedule known to both sender and receiver. This shared sense of time is used as security material, both for header compression and for replay protection.

DL clock sources may be configured to periodically transmit DL advertisements embedded in the DL's DAUX sub-header. Each such advertisement provides a TAI time reference for the DL subnet. All standard-compliant devices that receive an advertisement are assumed to be capable of participating in time-synchronized communication with the advertising device. DL clock recipients with relatively imprecise clocks may have a limited capability to communicate only with routers that use slow hopping.

MAC layer authentication requires shared security keys, shared time sense, and knowledge of a neighbor's EUI-64. This information is acquired stepwise during the join process, with security keys provided at the end. AES-128 with a well-known security key is used by the DL during the join process in order to provide enhanced integrity checking (but not security). This is described in more detail in 7.4.

In slotted hopping, both sender and receiver share an intrinsic sense of which timeslot is being used; otherwise, the devices would not be able to communicate. Every timeslot has a scheduled start time that is known by both sender and receiver.

In slow hopping, a device with an inaccurate sense of DL time will nominally transmit in a particular timeslot, based on its own sense of time. However, such a device is permitted to miss its target, and may actually transmit in any timeslot within the slow hopping period. Therefore, unicast DPDUs that are transmitted using a slow hopping superframe shall add an extra octet to the DL header to indicate which timeslot within the slow hopping period was intended. This allows the receiving device to reconstruct timeslot information from the

transmitting device, to apply time correction across timeslots, to validate the accuracy of the transmitter's clock, and to use the timeslots scheduled start time as security material.

For security purposes, a timeslot's scheduled start time (not the start time of a DPDUs) is passed to the security sub-layer for use in DL-related security operations. See 9.1.11. In most cases, the timeslot of the DPDUs and the timeslot of the acknowledgement is the same. There is one exception, which is in slow hopping wherein the clock of one of the devices has drifted into a different timeslot. In that case, the slow hopping timeslot offset of the acknowledging device shall be included in the acknowledgement's DMXHR (Table 117) to identify unambiguously the time that is used for security operations, even if the acknowledging device is not acting as a DL clock source for the acknowledgement recipient. Absence of the slow hopping timeslot offset in the acknowledgement implies that the timeslot of the DPDUs and acknowledgement are the same.

### 9.1.9.3 Pairwise time synchronization

#### 9.1.9.3.1 General

Clock updates are propagated through the DL subnet in the course of normal communications within a timeslot or slow hopping period. These building blocks are leveraged by the system manager to arrange the propagation of network time.

DL clock sources use three general mechanisms to propagate clock updates to their neighbors.

- A DL clock source originates a DPDUs that includes an advertisement. The time is conveyed based on when the DPDUs is transmitted.
- A DL clock source acknowledges a received DPDUs in a timeslot. The time is conveyed by measuring when the DL clock source detects the start of the DPDUs and echoing the result of this measurement in the acknowledgement.
- A DL clock source acknowledges a DPDUs within a slow hopping period. The process is similar to acknowledgement within a timeslot, with the addition of an octet to identify the timeslot uniquely if needed.

The standard relies on stable clocks, particularly in field routers, for reliable DL subnet operation. For DL clock stability requirements, see Table 485.

A DL clock recipient maintains a list of valid neighboring DL clock sources. If it receives a clock update from a designated DL clock source, it uses the data to update its clock.

When a device discovers a DL subnet, it acquires the network's TAI time and uses that reference for subsequent communication. The device shall then periodically re-synchronize its internal clock to the subnet clock. These re-synchronization operations are incremental, based on offsets into a timeslot with a scheduled time reference known to both DL clock source and DL clock recipient.

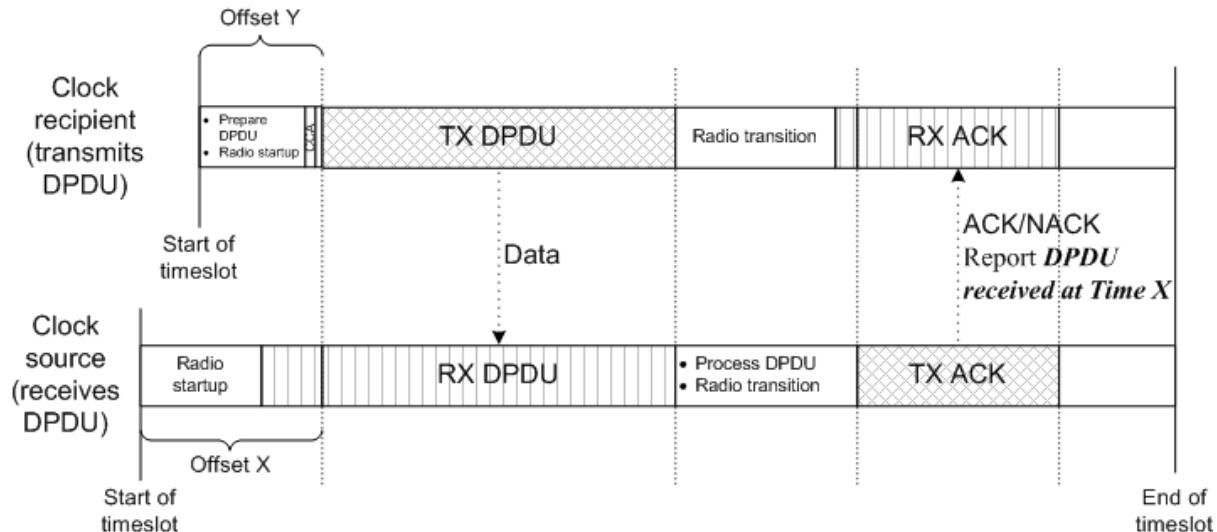
Whenever a device interacts with one of its designated DL clock sources, it receives updated clock information. Clock adjustments are included in acknowledgements, thus conveying time information to the recipient. Clock updates are also included in the DAUX sub-header of advertisement DPDUs originating from DL clock sources.

Thus, a device may receive clock updates as a by-product of routing data through a DL clock source. Alternatively, if a device needs more frequent clock updates, it may be configured to receive clock updates by enabling its receiver to operate at times coinciding with periodic scheduled DPDUs from DL clock sources that include the DAUX sub-header with an advertisement.

A device may acquire a clock update by sending a DPDUs with a zero-length DSDU to a DL clock source. A DL clock source shall acknowledge such a DPDUs and then discard it.

### 9.1.9.3.2 Clock source acknowledges receipt of a data link layer protocol data unit within a timeslot

When a device transmits a unicast DPDU (see Figure 81) to a DL clock source, it may receive a clock update in the ACK/NACK DPDU, as shown in Figure 78.



**Figure 78 – Clock source acknowledges receipt of DPDU**

The ACK/NACK DPDU from the DL clock source includes the start time of the original transmission; reported as a time offset (with microsecond precision, in units of  $2^{-20}$  s) of the DPDU's start time (i.e., the start time of the PSDU) from the scheduled timeslot start time as measured by the DL clock source.

A given unicast transaction occurs on a single channel, with the DPDU and acknowledgement both transmitted on the same channel.

This clock synchronization information is targeted at the device that originated the DPDU. While other devices may overhear and update their clocks based on the same information, the design is not optimized for that usage.

While a device's internal details of clock synchronization are not specified by this standard, the transaction is intended to support a clock synchronization process that operates generally as follows:

- The DL clock recipient sends a DPDU to the DL clock source and records that it sent the DPDU at offset Y.
- The DL clock source receives the DPDU at exactly the same time and records that it received the DPDU at offset X.
- The diagram shows the case where the DL clock recipient's clock is running faster than the DL clock source. In that case,  $X > Y$ . If the DL clock source is running faster,  $X < Y$ . The time of the DL clock source is assumed to be correct.
- In the ACK, the DL clock source reports that it received the DPDU at offset X.
- The DL clock recipient applies a time correction of  $Y - X$ .

The result shown in the figure is that the timeslots start at different times, but end at the same time. An actual implementation may delay application of time corrections, such as by adjusting the clock gradually. See 9.1.9.2.2.

### 9.1.9.3.3 Clock source originates a data link layer protocol data unit that includes an advertisement

A DL clock source is often the originator of a DPDU. This is generally the case when a DL clock source transmits a scheduled advertisement that includes the TAI time. The payload of

the DPDU may be unrelated to time synchronization; the information is contained within the DAUX sub-header, so that it may be overheard by any of the DL clock sources neighbors.

Multiple devices may be configured to enable their receivers simultaneously in anticipation of a scheduled DPDU conveying an advertisement.

NOTE This standard requires that a DL clock source be capable of precisely controlling when advertisements are transmitted, with a precision of  $\pm 96 \mu\text{s}$  (3 octets) in reference to its internal DL clock. See 9.4.3.3.1.

#### **9.1.9.3.4 Clock source acknowledges a data link layer protocol data unit within a slow hopping period**

As noted previously, DL clock sources use acknowledgements to report time with microsecond resolution ( $2^{-20} \text{ s}$ ) since the start of a timeslot. Timeslot identity is assumed to be unambiguously known by both sender and recipient; thus, only the offset is needed in clock updates.

However, within slow-hopping periods, a DL clock recipient's internal clock may have drifted tens or hundreds of milliseconds or more, and a shared timeslot reference might not be implicit. Therefore, an extra octet, identifying a specific timeslot, is added to DL headers within slow-hopping periods. The initial timeslot within a slow hopping period is defined as having an offset of zero.

Within slow-hopping periods, devices with an accurate sense of time should operate within timeslot boundaries. However, devices without an accurate sense of time might not be capable of respecting timeslot boundaries, and might not even know the timeslot they are actually using. A timeslot offset in the DL header allows the receiver to reconstruct the sender's time sense and vice versa. (See 9.3.3.3.)

#### **9.1.9.3.5 Auditing the quality of a neighbor's clock**

When a device joins the network, it reports its clock stability specifications to the system management function. In their normal interactions with these devices, DL clock sources and DL clock recipients can be configured by the system to collect diagnostics, on a per-neighbor basis, to audit these nominal specifications. (See 9.4.3.9.)

#### **9.1.9.3.6 Discontinuous clock adjustments**

Under some conditions, it may be necessary for the system manager to adjust all of its device's clocks by tens or hundreds of milliseconds or more, for example, when two DL subnets are being joined. The system manager may schedule a discontinuous clock adjustment to occur at a particular TAI time, by setting the dlmo.TaiAdjust attribute on each of its devices. At the designated time, all devices shall adjust their clocks forward or backward by the specified amount of time.

There are some system impacts of a clock adjustment that should be considered whenever this feature is used.

- The security model in this standard does not allow time to run backward. Time is used in the security nonce, which can never be repeated in any standard communication layer. Consequently, there shall be an interruption in service, equal to the magnitude of the adjustment plus at least one timeslot, if time is adjusted to an earlier time.
- Phased reports will shift in time by the amount of the clock adjustment, unless corresponding contracts are revised in tandem with the clock adjustment.
- Devices that do not receive the time correction before the cutover time may lose synchronization with subnet operation, and may need to re-discover their neighbors.

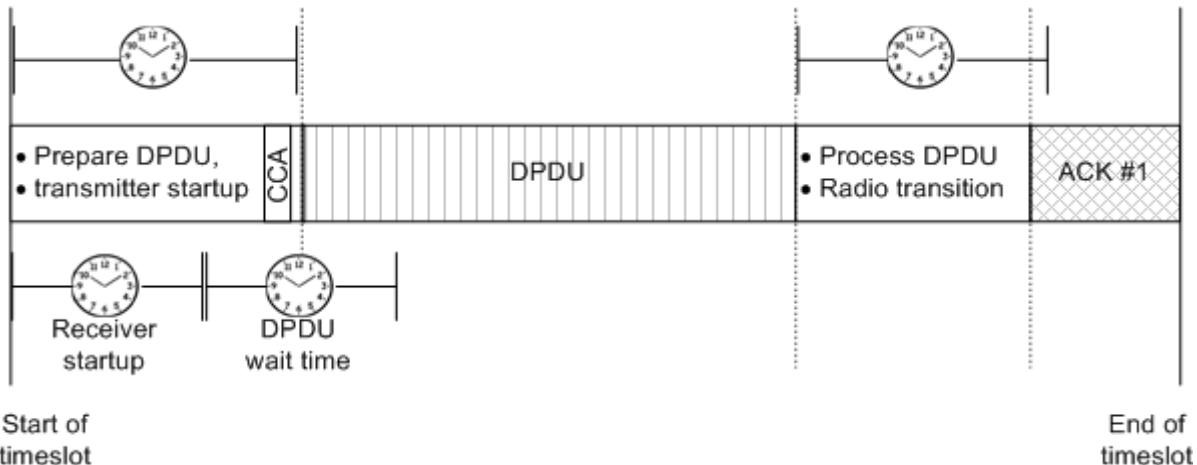
#### **9.1.9.4 Transactions within timeslot templates**

##### **9.1.9.4.1 General**

Transaction timing within a DL timeslot is specified by timeslot templates. The standard defines default timeslot templates that are needed by the field device in its interactions with a wireless provisioning device. Additional timeslot templates may be added by the system

manager during or following the join process. (See 9.4.3.3 for more information on timeslot templates.)

Figure 79 illustrates some aspects that are addressed by DL timeslot templates.



**Figure 79 – Transaction timing attributes**

As shown in Figure 79, timeslot templates define the timing for operations such as DPDUs reception wait time, and the turnaround time (DPDUs reception processing and radio transition) between receiving a DPDUs and transmitting an acknowledgement DPDUs.

Generally, there are two types of transaction templates: transmit and receive. A transmit template specifies a time range to begin DPDUs transmission, whether to check the channel before transmission, and the amount of time to wait for an acknowledgement. A receive template specifies when an incoming DPDUs can begin arriving, when to timeout if no DPDUs is seen, and timing attributes for the acknowledgement.

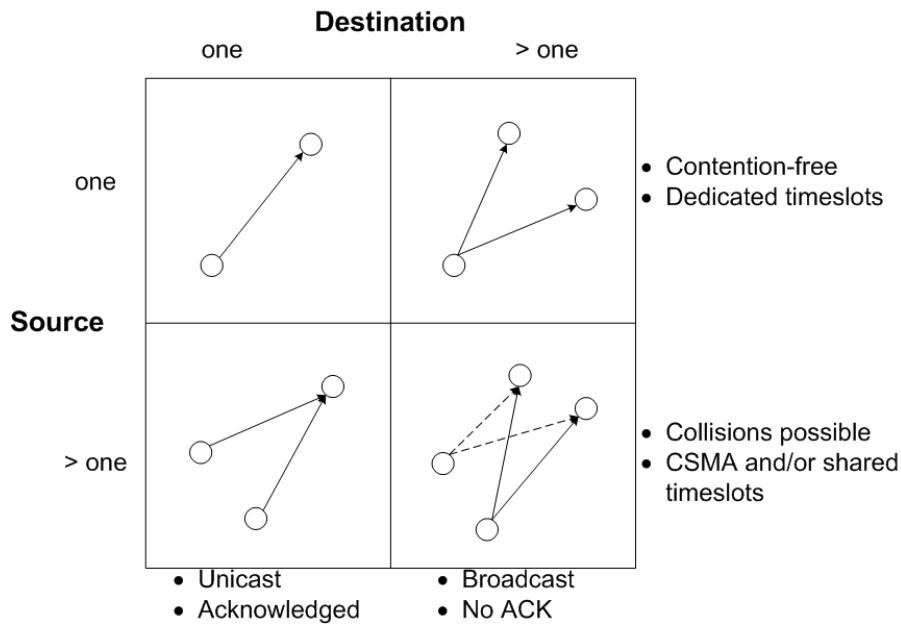
Default timeslot templates cover baseline transactions needed to interact with a provisioning device, and used during the join process. Default templates may also be used by joined devices for general transactions. Additional templates may be provisioned into the device to accommodate features such as:

- Carrier sense multiple access with collision avoidance (CSMA-CA) periods at the start of a timeslot. (See 9.1.9.4.8.)
- Extended ACK/NACK wait times. (See 9.1.9.4.6.)

By convention in this standard, timeslot template timing is specified based on the start and end times of DPDUs. PPDU timing, dependent on the details of the physical layer that contains the DPDUs, can be inferred from DPDUs times. (See 9.4.3.3.)

#### 9.1.9.4.2 Data link layer transaction overview

As shown in Figure 80, the DL supports both unicast and broadcast transactions.



**Figure 80 – Dedicated and shared transaction timeslots**

Unicast transactions, indicated in the left half of Figure 80, may use dedicated timeslots, for example, when a device reports repetitively on a schedule. Duocast is a variant of unicast, wherein a second receiver is scheduled to overhear the DPDU and provides a second acknowledgement. Duocast is shown graphically in Figure 84.

A positive acknowledgement (ACK) indicates that the receiver has successfully received the DPDU and that the transmitter can mark the transaction as complete. Unicast and duocast transactions require that a link-layer acknowledgement be sent in response.

Broadcast transactions, indicated in the right half of Figure 80, may also use dedicated timeslots, such as for scheduled advertisements. Broadcast transactions cannot use a link-layer acknowledgement.

As shown in the lower left quadrant of Figure 80, unicast and duocast transactions may use shared timeslots, such as within slow-hopping periods. Shared timeslots are commonly used for retries, join requests, exception reporting, and burst traffic.

As shown in the lower right quadrant of Figure 80, broadcast transactions such as solicitations may also use shared timeslots.

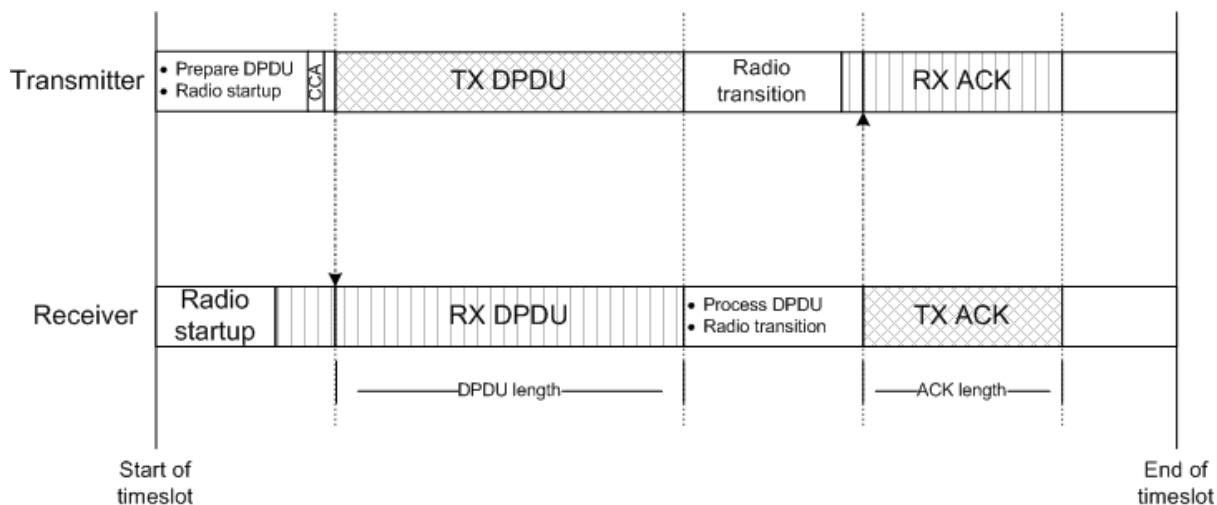
The term contention-free in Figure 80 is relevant only within the scope of DL subnet timeslot allocation. If two subnets, or devices within the same subnet, are allocating timeslots in an uncoordinated fashion, then access is not contention-free. Likewise, other users of the 2,4 GHz spectrum, such as WiFi, Bluetooth, ZigBee, or WirelessHART, potentially may also interfere. Improved coexistence with these uncoordinated systems is achieved by using CCA (see 9.1.9.4.3) to check the channel before transmission.

The use of broadcast in the standard is limited to DL-layer operations—advertisements, solicitations, and duocast. Advertisements and solicitations, used for subnet discovery, are described in 9.1.13. Duocast is described in 9.1.9.4.7.

NOTE Broadcast as described here does not use broadcast MPDUs defined in the IEEE standard. See 9.1.5. Broadcast and multicast, as application layer services, are not supported in this version of the standard.

#### 9.1.9.4.3 Unicast transaction

Figure 81 illustrates a unicast transaction.



**Figure 81 – Unicast transaction**

Before a DL transmits a DPDPU in a timeslot, it prepares the DPDPU for transmission, generally with link layer security. Prior to transmission, the DL is normally configured to perform a clear channel assessment (CCA) of the radio space.

CCA shall be implemented as described in IEEE Std 802.15.4. IEEE specifies a detection period of 8 symbols. CCA shall be performed as configured in the timeslot template field `dlmo.TsTemplate[].CCAmode` (see Table 163), with the choices being:

0. No CCA;
1. CCA Mode 1 (energy above threshold);
2. CCA Mode 2 (carrier sense only);
3. CCA Mode 3 (carrier sense with energy above threshold).

Compliance to IEEE Std 802.15.4 requires that at least one of the CCA modes be supported. If a non-zero value for `dlmo.TsTemplate[].CCAmode` is selected and the selected mode is not supported by the device, the DL may choose a different non-zero CCA mode that is supported by the device. CCA modes supported by the device are indicated in `dlmo.DeviceCapability.SupportedCCAmodes`; see 9.4.2.23.

If CCA reports a busy medium, the transmission transaction shall be aborted.

Use of CCA as defined by IEEE is not intended to exclude additional CCA checks that might be supported by advanced devices. For example, if a backbone router is capable of detecting IEEE 802.11 modulation, the DL may reasonably leverage this capability to detect and report a busy medium.

CCA should be complete 192 µs prior to the start of the physical layer header.

If the DPDPU requires an acknowledgement (ACK), the transmitting DL enables its receiver after the transmission is completed. If an ACK is received, the transmitted DPDPU is deleted from the DL's transmit queue.

When a DL is scheduled to receive a DPDPU, it enables its radio receiver and waits for the incoming PPDU. If it detects a valid IEEE Std 802.15.4 SHR and PHR, it continues to receive the entire PPDU. It then processes the contained DPDPU (including link layer MIC authentication) and determines if the DPDPU requires an acknowledgement. To send an acknowledgement (ACK or NACK), a receiving DL enables its radio transmitter and sends the ACK/NACK within the same timeslot.

The acknowledgement time window is defined by the timeslot template. If there is a substantial delay between the end of the DPDU and the scheduled start of the acknowledgement, an implementation may sensibly disable its receiver during the delay.

#### 9.1.9.4.4 Negative acknowledgements

A DL shall respond to a unicast DPDU with a negative acknowledgement (NACK) when it cannot accept the DPDU at the time, but has received it with no errors. Time synchronization information may be included with NACKs. Similarly, a NACK ensures that RF statistics correctly log a clean transmission. A NACK can be used to exert back-pressure as a simple flow control mechanism.

The DL supports two types of NACKs, NACK0 and NACK1. A NACK0 is intended to indicate resource limitations in the router, while a NACK1 is intended to signal downstream connectivity problems in the DL subnet.

The DL shall respond to a unicast DPDU with a NACK0 when it correctly receives the DPDU but cannot accept it due to lack of capacity in its message queue. See 9.1.8.5 for a discussion of the DL message queue. A DL may also respond with a NACK0 when configured in excess of its forwarding capability (ForwardRate) as described in 9.4.2.23.

The DL may respond to a unicast DPDU with a NACK1 to apply back pressure in the event of lost downstream connectivity. For example, when the DL loses downstream connectivity to all of its neighbors following Graph #12, and then receives a DPDU also following Graph #12, the DL may sensibly respond with a NACK1 rather than accepting more DPDUs that it is unlikely to be able to forward.

When a DL receives a NACK0 or a NACK1 from a neighbor, it shall backoff by not transmitting DPDUs to that neighbor for a period of  $\text{dlmo.NackBackoffDur}$ . The NACK backoff does not include DPDUs without a payload, thus allowing the DL to poll a DL clock source for a time update even though the neighbor is not accepting DPDUs at this time.

As described in 9.1.9.2.2, if a DL clock repeater's clock has expired and it is polled for a time update, it should respond with a NACK1.

#### 9.1.9.4.5 Explicit congestion notification

The standard supports explicit congestion notification (ECN) as described in IETF RFC 3168. ECN provides a mechanism for a router to affect application layer flow control,

As described in 12.12.4.2.3, there is a limited number of data source requests that can be simultaneously awaiting response from a data sink. Flow control at the data source operates by incrementally increasing the limit on outstanding requests, based on receipt of timely data sink acknowledgements.

ECN provides a mechanism whereby flow control can be effective without driving the network to the point of failure. Any router along the path from data source to data sink may set ECN to indicate that the router is nearing its capacity. When the data sink receives the ECN, it echoes it back to the data source, which then accounts for the ECN in its flow control logic.

An ECN indicator in the DL header may be set by any field router that is experiencing congestion, following the guidelines in IETF RFC 3168, as a signal to a data source that it should apply flow control to reduce its use of network resources. This ECN indicator is propagated to the data sink, such as a gateway, and eventually works its way back to the data source through a transport layer acknowledgement.

In addition, the DL provides a special type of acknowledgement called ACK/ECN. A DL receiving an ACK/ECN treats it as a normal DL acknowledgement. In addition, the device may treat the ACK/ECN as an early indication that the data sink's acknowledgement will include an ECN.

Use of the ACK/ECN should be limited to DPDUs with a priority of seven (7) or less, corresponding to best effort queued and real time sequential flows.

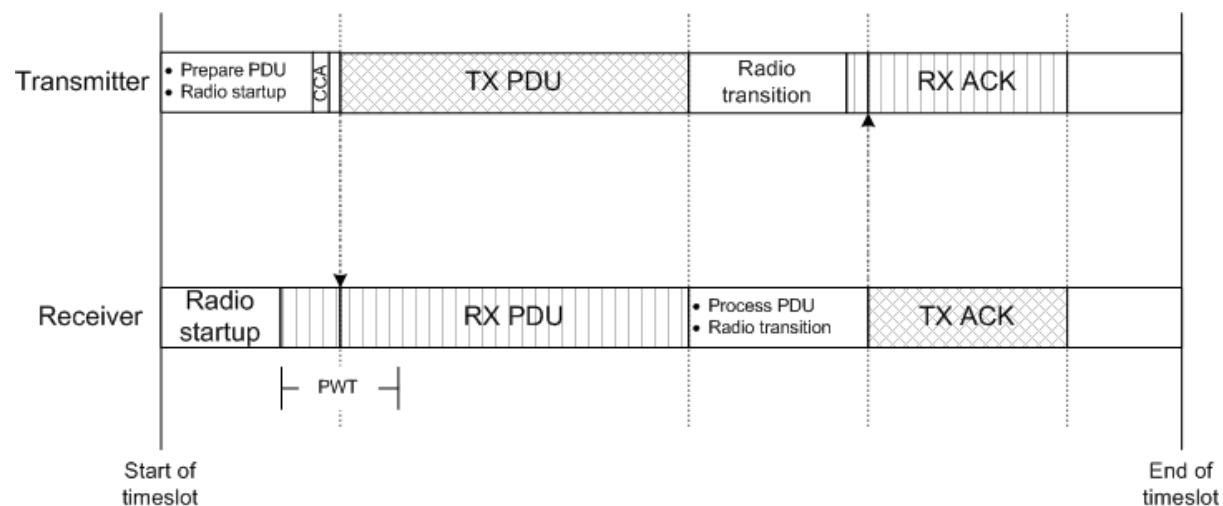
#### 9.1.9.4.6 Data link layer protocol data unit wait times

The clock times of transmitting and receiving devices are rarely in perfect synchronization. Therefore, a transmitting device is unlikely to transmit a PDU (protocol data unit) at exactly the time that a receiving device expects it. If a PDU is transmitted too early, the receiver might not yet be enabled. If a PDU is transmitted too late, the receiving device may have disabled its radio in order to save energy. PDU wait time (PWT), is the time period when the receiver is expected to listen for incoming PDUs. A particular degree of timing accuracy between the devices is implicit in the system manager's selection of PWT.

Devices compliant with this standard accommodate configurable PWTs and configurable timeslot durations. PWT is not directly specified in the standard, but can be inferred within timeslot templates from the earliest and latest times that PDU reception begins. See Table 161.

**NOTE 1** This tutorial does not make a distinction between PPDU and DPDU timings. As specified in 9.4.3.3, the DL follows a convention of specifying timeslot timing in reference to the DPDU (PSDU), not the PPDU. In implementations based on IEEE Std 802.15.4 (2.4 GHz), PPDU header detection involves receipt of a preamble, start frame delimiter (SFD), and frame length, for a total PPDU header of 192  $\mu$ s (6 octets) before the DPDU begins. Similarly, radio transmission of a PPDU begins 192  $\mu$ s prior to the nominal DPDU start time.

PDU wait times in a unicast PDU are illustrated in Figure 82. The same principles apply to other types of PDUs.



**Figure 82 – PDU wait time (PWT)**

The duration of the PWT is configured by the system manager accounting for the intrinsic stability of transmitting and receiving devices clocks, and the guaranteed maximum time between clock updates.

For example, if transmitting devices are accurate to  $\pm 1$  ms relative to the receiver within the clock expiration period (dlmo.ClockExpire), the receiver's PWT should be 2 ms long. A less accurate transmitter will require a longer receiver PWT or a shorter clock expiration period. (In this example, the radio's listening time should be slightly longer than the 2 ms PWT, to account for the PPDU's SHR and PHR durations.)

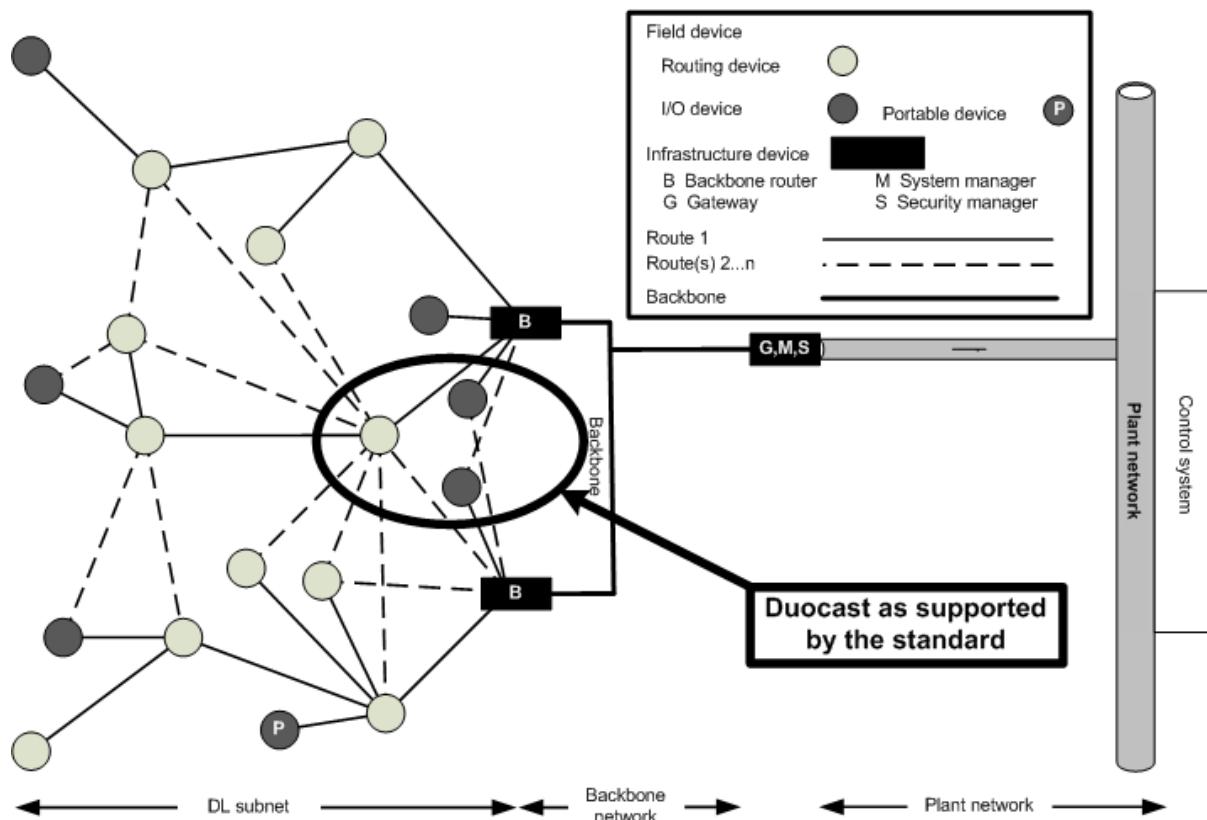
If the receiver does not begin receiving the expected PDU by the end of its PWT, the receiver is permitted to disable its radio for the duration of the timeslot.

Ten ms timeslot durations have a timing budget that can allocate about 2 ms to PWT. For a longer PWT, either other allocations have to be adjusted or the timeslot duration has to be increased. For example, if the network supports 15 ppm devices that sleep for up to two minutes, timing errors of about  $\pm 2$  ms may accumulate between reports. This can be accommodated by increasing the PWT from (for example) 2 ms to 4 ms, with a corresponding increase in timeslot duration and receiver energy consumption.

NOTE 2 Devices with infrequent reporting intervals may be configured to check the network periodically for incoming DPDUs. These devices may receive clock corrections through the DAUX sub-header as part of the same process.

#### 9.1.9.4.7 Duocast transaction

Duocast is a variant of unicast, wherein a second receiver is scheduled to overhear the DPDUs and provides a second acknowledgement. Duocast provides support for latency-controlled access to a backbone with a high probability of first-try success. Duocast transactions are defined by this standard only for field devices with links to two or more infrastructure devices, particularly to backbone routers, as shown in Figure 83.



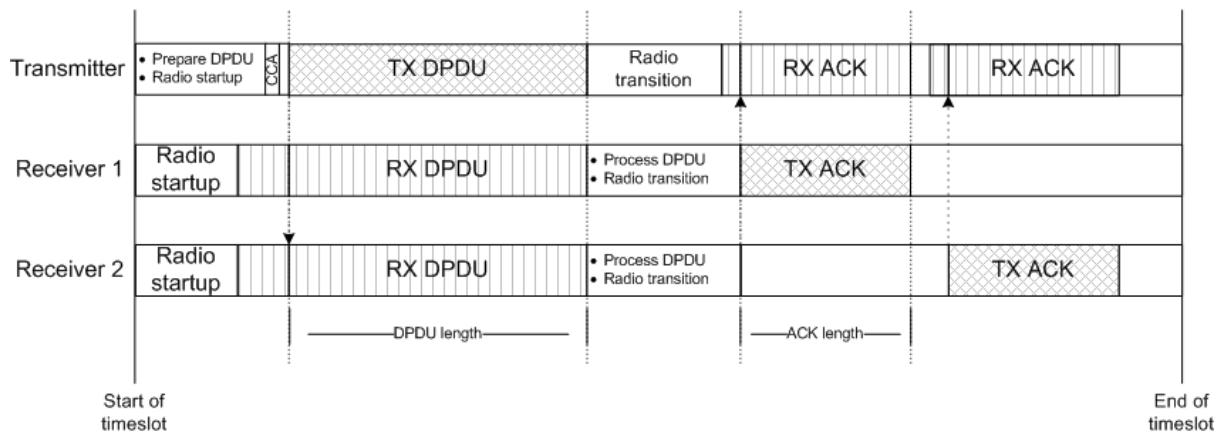
**Figure 83 – Duocast support in the standard**

Duocast support in the standard is limited to the case where a field device is within range of two infrastructure devices, such as two backbone routers. Devices receiving the duocast acknowledgements, circled in Figure 83, need to be configured with timeslot templates with an extended listening window for the additional acknowledgement(s). Duocast support typically involves increased timeslot duration of approximately 1 ms to 2 ms, as configured by the system manager.

NOTE Transmission of the duocast response is not specified by this standard. Coordination of the duocast response may involve back channel coordination between the responding infrastructure devices.

Duocast timeslots may be scheduled in conjunction with available digital bandwidth for a fast retry on a second channel, as shown in Figure 66.

Figure 84 illustrates a transaction involving duocast transmission and reception.



**Figure 84 – Duocast transaction**

In the example in Figure 84, the DPDU is addressed to Receiver 1, the primary recipient, and is overheard by Receiver 2, a secondary recipient. For duocast transactions, the destination address of the DPDU is set to that of the primary recipient (Receiver 1 in Figure 84), and an acknowledgement is expected from at least one recipient, during the same timeslot. As illustrated in Figure 84, the primary recipient transmits an ACK upon receipt of the DPDU. The secondary recipient also transmits an ACK, but after a delay to allow time for the first ACK to complete. The ACK from the secondary recipient includes its own address in the source address field of the MHR.

If an ACK is received from the first recipient, the transmit device is not required to expend energy receiving and processing the second ACK. The transmitter should periodically verify an ACK from each recipient to confirm the continuing availability of both receivers at all times. If an ACK is received from either recipient, the transaction is complete and the DPDU is deleted from the DL message queue.

As described in 9.3.4, a duocast acknowledgement from a secondary recipient includes its own address field in the source address field of the MHR. This enables the transmitter to identify the acknowledgement's source.

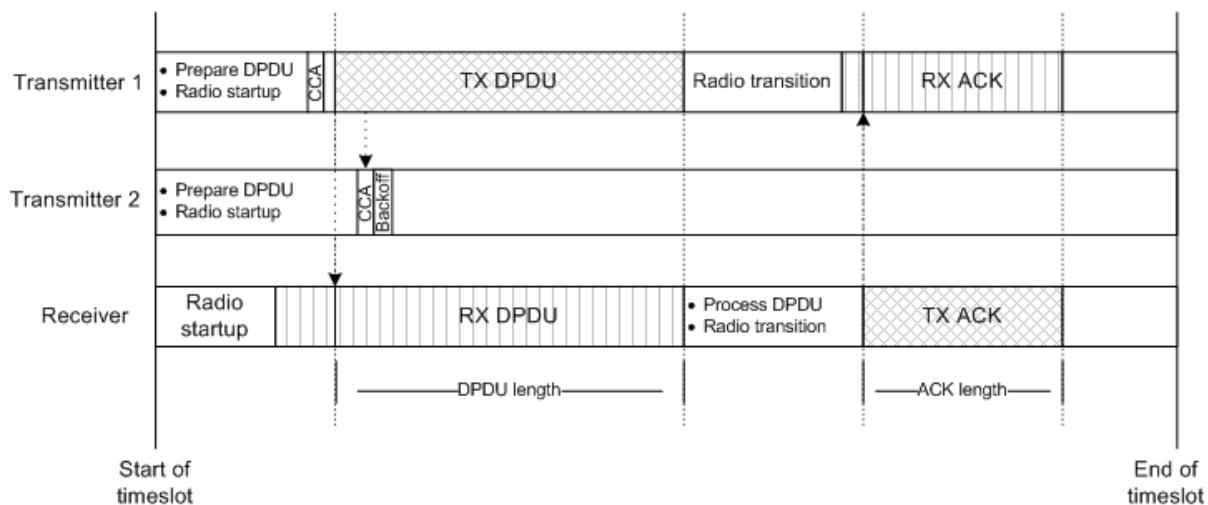
Duocast is not limited to two acknowledgements; more may be accommodated by expanding the time window for more acknowledgements.

#### 9.1.9.4.8 Shared timeslots with carrier sense multiple access with collision avoidance

Unicast transactions may occur in timeslots that are dedicated to a specific link. Alternatively, shared timeslots may be designated to provide bandwidth on demand to a collection of devices. Shared timeslots are typically configured to transmit only near the start of the timeslot.

Timeslot templates specify transmission time as a range as per 9.4.3.3. At a minimum, the transmit time shall be configured to a range of at least 192 µs, thereby allowing for ±96 µs ( $\pm 6$  PHY symbol periods) of jitter that is permissible in a transmitter. If the transmission time range is configured to be larger than 200 µs, the device shall select a randomized time within the range to begin its transmission, making reasonable accommodation for the device's actual transmission jitter characteristics.

Figure 85 illustrates the use of a shared timeslot with CSMA-CA.



**Figure 85 – Shared timeslots with CSMA-CA**

In the example in Figure 85, two devices are contending for use of the channel in a shared timeslot. This approach is used for all shared slots, configurable to be used in various ways.

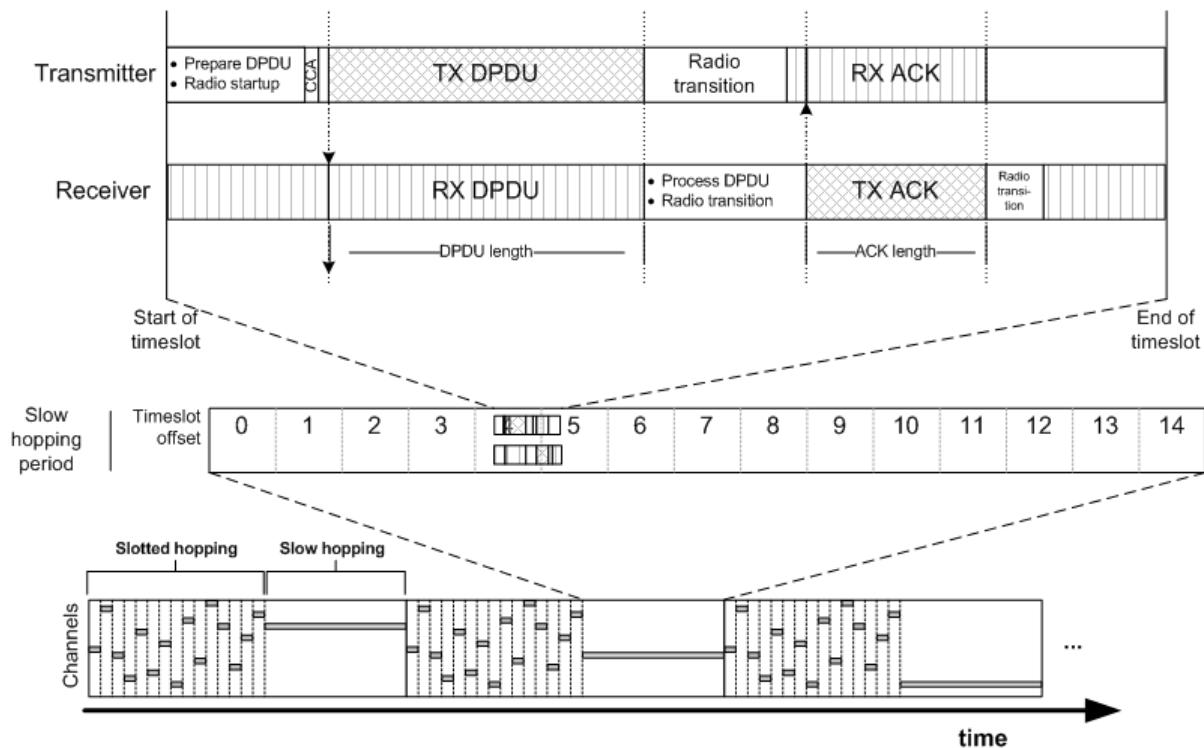
Priorities within a shared timeslot may be managed by configuring devices with different timeslot templates. A device with a high-priority DPDUs, such as a retry for a failed duocast transaction, may be configured to transmit its DPDUs as early as possible within the timeslot. A device with less critical requirements may be configured to delay its transmission to slightly later in the timeslot, such as 2 ms later. If another device has already claimed the timeslot, as shown in Figure 85, the CCA of the delayed device might (or might not) detect that the channel is in use and consequently defer its transmission to another timeslot.

Use of CSMA-CA within shared timeslots may involve configurations with longer timeslots and longer DPDUs wait times, and use of more receiver energy.

#### 9.1.9.4.9 Transactions during slow-hopping periods

Some devices do not have sufficient timing accuracy to communicate within short timeslots and thus may need to use slow-hopping periods for their communication. Slow-hopping periods are simply a set of concatenated timeslots on the same channel, wherein the receiver runs its radio continuously and the transmitter is not required to respect timeslot boundaries.

As shown in Figure 86, a transaction during a slow hopping period is very similar to a unicast transaction in a shared timeslot, except DPDUs can be transmitted at any point during the slow hopping period. Transmitting devices target the beginning of a specific timeslot within a slow hopping period, based on the transmitters own sense of time, which is not required to be very accurate. Transmitting devices use CCA to check the channel before transmission. In the absence of higher priority operations (such as forwarding DPDUs), the receiver hosting the slow hopping period runs its radio receiver continuously except when responding to DPDUs that it receives.



**Figure 86 – Transaction during slow-hopping periods**

Devices can aim for any timeslot within a slow hopping period, with one major caveat: scheduled DPDUs timeslot time is required to increase with each transaction.

Devices with accurate clocks should respect timeslot boundaries within slow-hopping periods. If all devices within slow-hopping periods are well behaved, with accurate clocks, performance is approximately comparable to slotted ALOHA.

A device with inaccurate timing will nominally transmit near the start of a particular timeslot, based on its own sense of time. However, such a device may actually initiate transmission in any phase of any timeslot within the slow hopping period.

For example, a device with a  $\pm 50$  ppm clock, due to uncompensated environmental fluctuations, may sleep for 3 minutes between transactions, corresponding to a clock drift of about  $\pm 9$  ms. A device of this type might wait for a scheduled advertisement to get itself resynchronized prior to transmission. Alternatively, it might transmit during a slow hopping period (if available) and receive time synchronization in the acknowledgement. Continuing with this example, a device with  $\pm 9$  ms accuracy would select one of the slots within the slow hopping period, and transmit using the appropriate link template. The device would nominally be transmitting in a particular timeslot, based on the device's own sense of time, but may actually be transmitting in a different timeslot.

**NOTE** In this example, the device should not attempt to transmit in the first or last timeslot in the slow hopping period, because it may actually transmit outside of the available time range. It is the device designer's responsibility to avoid selecting timeslots that are inconsistent with the device's time keeping capabilities, accounting for the device's own uncalibrated clock drift in combination with the 10 ppm clock drift allowed for a neighboring router or the 100 ppm clock drift allowed for a neighboring I/O device.

Unicast DPDUs in slow hopping periods shall add an extra octet to the DL header to indicate which timeslot offset within the slow hopping period was intended. This enables the receiving device to reconstruct timeslot information from the transmitting device, to apply time correction across timeslots, to validate the accuracy of the transmitter's clock, and to use the scheduled timeslot start time as security material for message authentication. (This is specified in the data frame media access control sub-headers, DMXHR; see 9.3.3.4.) If necessary, a corrected timeslot offset is provided in the acknowledgement (see 9.3.4). Unambiguous shared timeslot identification is needed for both sender and receiver to authenticate the DPDUs and to resynchronize time. Even if the transmitting device has a highly

accurate sense of time, the receiver might not; therefore the slow hopping offset octet is required for all DPDUs using a slow hopping superframe.

The initial timeslot within a slow hopping period is defined as having an offset of zero.

### **9.1.10 Data link layer subnet addressing**

#### **9.1.10.1 Address types**

16-bit addressing shall always be used within a DL subnet, with the exception that EUI-64 addresses are used in a limited way during the join process.

Every device in a DL subnet is identified in three ways:

- Each DL subnet device has an EUI-64 identifier that is presumed to be unique.
- Each device compliant with this standard shall be assigned at least one 128-bit network layer address when it joins the network. However, at the DL level, only the 16-bit alias for this address (described next) shall be used.
- Each device that is accessible through a DL subnet has a subnet-unique 16-bit DL address, which is an alias for its 128-bit network layer address. The scope of any 16-bit DL alias address shall be limited to a particular DL subnet.

The EUI-64 shall be used as a new devices address for immediate neighbor addressing prior to and during the join process. Once a device has joined the network and received its network address, it shall be addressed by either its 128-bit network layer address or a 16-bit DL alias for that address.

The EUI-64 is also used in the MAC sub-layer security nonce. Whenever a 16-bit DL address is used by the DL for a unicast DPDUs or ACK, the recipient device needs a priori knowledge of the corresponding EUI-64 addresses. This neighbor information shall be provided by the system manager as part of the link establishment process. An exception is made for a new device that is communicating with a neighbor that has advertised its 16-bit address. In that case, the DL shall acquire the EUI-64 of the advertising device by sending the neighbor a DPDUs without a DSDU payload, and requesting the EUI-64 in the acknowledgement as described in 9.3.3.3. Security key policy in that case shall be K\_global with DMIC-32, the same as for other DPDUs involving a 64-bit address. (See 9.1.11 for discussion of the relationship between DL and security layers.)

When a 16-bit DL address refers to a device within a DL subnet, the 16-bit DL address always corresponds to the 16-bit MAC address for the device. When a DPDUs contains a 16-bit DL address that refers to a device not within the scope of the DL subnet, the DPDUs is routed to a backbone router or system manager, which in turn maps the 16-bit DL address to its 128-bit network layer address counterpart.

Most DPDUs include two 16-bit source addresses and two 16-bit destination addresses. One pair of source/destination addresses is for next-hop addressing at the MAC level. A second pair of source/destination 16-bit alias addresses is within the DADDR sub-header, indicating the ultimate NL source and destination.

Routing is usually specified by graphs, not by addresses. However, when source routing is used within a DL subnet, 16-bit addressing is used. Again, the one exception is that EUI-64 addressing is used during the join process for communication with an immediate neighbor on the DL subnet.

#### **9.1.10.2 Subnet identifier and uniqueness of 16-bit data link layer addresses**

The scope of a 16-bit DL address is a DL subnet. A neighboring subnet may use the same 16-bit DL address as an alias for a different device. Different DL subnets may use different 16-bit aliases to refer to the same backbone device.

Each DL subnet has a 16-bit subnet identifier (dlmo.SubnetID; see 9.4.2.1), which has the same value as the PAN ID found in the IEEE Std 802.15.4 header. Within the scope of a network, each DL subnet shall have a unique dlmo.SubnetID. However, different networks are not necessarily coordinated, so dlmo.SubnetIDs across co-located standard-compliant

networks are not guaranteed to be unique. Thus, it is possible, but unlikely, that a DPDUs from a different standard-compliant network will be received with what appears to be a valid 16-bit DL address and PAN ID. The DL relies on the security sub-layer to discard such DPDUs due to security key mismatch.

SubnetID=0 shall never be used as a subnet ID in this standard, except as an internal indicator that the DL is not participating in a subnet. SubnetID=1 shall be used exclusively to identify provisioning networks (see 14.1).

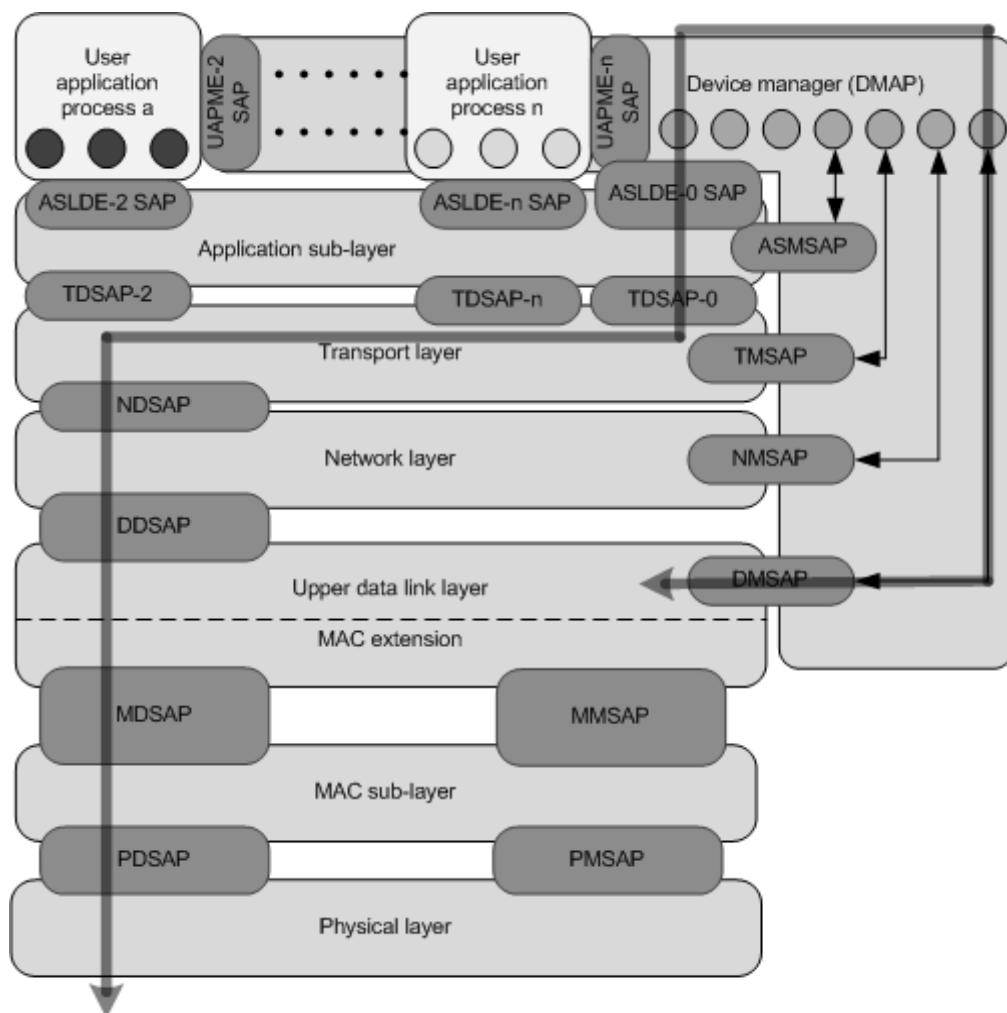
### 9.1.11 Data link layer management service

#### 9.1.11.1 General

Management messages to a field devices DL are fully secured at the application layer, using the end-to-end relationship between the system/security manager and the device's DMAP.

The IEEE Std 802.15.4 MAC is not accessed directly by the DMAP, but is instead configured indirectly through the DMSAP. This isolates the rest of this standard from evolutionary changes to IEEE specifications, and also facilitates future adoption of alternative physical layer specifications (e.g., radios), possibly with alternative or enhanced MACs.

As shown in Figure 87, DL management commands generally flow through the full communication protocol suite defined by this standard.



**Figure 87 – DL management SAP flow through standard protocol suite**

The DL is configured by the DMAP through a DMSAP management interface. Most management SAPs are generic, simply reading and setting data structures within the DL. The data structures generally define how the DL operates its state machine. The DMAP

communicates with the system manager through the application sub-layer, using end-to-end security.

For information on the general handling of standard management objects, see 6.2.5 and 6.2.6.

### **9.1.11.2 Management attributes and indexed attributes**

DMSAPs involve the manipulation of the DL management object (DLMO). The DLMO includes a variety of attributes that are used to configure the DL and/or report its status.

Some DLMO attributes apply to specific values. For example, attribute dlmo.SubnetID provides the subnet ID for the DL subnet that the device has joined.

Some DLMO attributes can be visualized as tables with a collection of indexed rows. Each such attribute is specified as an indexed OctetString. For example, the DL includes the attribute dlmo.Neighbor, which is an indexed OctetString attribute containing a collection of neighboring devices. Each entry of the dlmo.Neighbor attribute includes a set of fields for each neighbor, such as an indicator of whether the neighbor is a DL clock source. Each entry of the dlmo.Neighbor table is uniquely identified by the neighbor's 16-bit DL address. Indexed OctetString attributes in the DL include:

- dlmo.Ch: Channel hopping patterns;
- dlmo.TsTemplate: Timeslot templates;
- dlmo.Neighbor: Device neighbors;
- dlmo.NeighborDiag: Diagnostics for device neighbors.
- dlmo.Graph: Graphs for routing;
- dlmo.Superframe: Superframes specifying common attributes for associated links;
- dlmo.Link: Links, each of which is associated with a superframe;
- dlmo.Route: Routes to be used in DROUT sub-headers.

Relationships among these attributes are described in 9.4.3.1.

DLMO attributes shall be maintained through the DMAP using methods that are described in Clause 9 and Clause 12. The ASL services Read and Write described in Table 271 provide a general framework for reading and writing DLMO attributes. Methods for writing attributes that are to become active at a TAI cutover time, as well as methods for reading and writing indexed OctetString attributes, are described in 9.5.

The format of the DLMO attributes is defined in the DL specification. When these objects are embedded within over-the-air messages, the specified formats shall be used. This is not intended to constrain the way that this data is stored within the DL, or to define how corresponding messages are conveyed across the device's internal DMSAP interface.

### **9.1.11.3 Management messages from immediate neighbors**

Any DL message may include a DAUX sub-header, for example carrying advertisement information from an immediate neighbor. The DAUX sub-header information is not propagated to higher layers of the communication protocol suite. In most cases, the content of the DAUX sub-header is intended for the recipients DLs management process, which in turn may use this information to configure the DL state machine. For example, the DAUX sub-header may include superframe definitions that are intended to be used by the device as a starting point in the join process and/or for neighbor discovery.

The DAUX sub-header is modeled as being instantly visible to the DMAP and immediately acted on if necessary. This aspect of the device's internal DMSAP interface is not standardized.

### **9.1.11.4 Multiple subnets**

In this standard, all DL management objects are specified to support only one active DL subnet at a time. All 16-bit DL addresses shall be unique within the scope of that single DL subnet. This is not intended to prevent a device from participating in multiple DL subnets

simultaneously. Multiple DL subnets might be modeled as multiple instances of the DL, but such operation is not specified by the standard.

If dlmo.SubnetID=0, the field device is not yet participating in a DL subnet.

### **9.1.11.5 Multiple radios**

DL SAPs assume that only one radio operates at a time on a given device. This is not intended to prevent proprietary implementations where multiple radios are used in parallel. Multiple radios might be modeled as multiple instances of the DL, but such operation is not specified by the standard.

### **9.1.12 Relationship between data link layer and security sub-layer**

The relationship between the DL and security sub-layers is described in 7.3.2. Careful review of that sub-clause is essential for anyone who wishes to fully understand DL operation.

Generally, the DL relies on the security sub-layer for integrity checking as well as security. All DPDUs include a DMIC that unambiguously validates that an MPDU originates from a compatible device, vs. another protocol using the same radio. The DMIC uses a non-secret security key during the join process; and then is normally configured to use a secret security key once the device is joined.

DPDUs and ACKs are expanded by the DL, by inserting the DPDUs DMIC into the ACK header, as a virtual field before being processed by the security sub-layer. This virtual field, which is not transmitted, is included in the ACK's DMIC calculation. Thus the DPDUs DMIC is essentially echoed in the ACK without actually transmitting the field. In this manner, the ACK is unambiguously connected to the corresponding DPDU.

### **9.1.13 Data link layer neighbor discovery**

#### **9.1.13.1 General**

Wireless subnets compliant with this standard are detected by the DL. A new device may hear advertisements from neighboring devices that have already joined a DL subnet of interest. This is called neighbor discovery. Following neighbor discovery, the device can join the network as described in 7.4.

The DL advertisement and neighbor discovery process are the building blocks that enable a device to find the 16-bit DL address and EUI-64 of a neighboring router and a set of scheduled links to communicate with that neighbor. That information is then used by the device's local management task to join the network.

Devices discover DL subnets through DL advertisements received from one or more advertising devices that periodically announce their presence. An advertising device is capable of acting as a proxy in the provisioning or joining process. An advertisement contains information that enables a new device's DL to send a join request to the advertising device and subsequently to receive a join response after some elapsed time.

After joining the network, the device receives advertisements from a series of neighbors in a DL subnet and builds internally a history of candidate neighbors that are in range. This list of candidates is reported to the system manager through the attribute dlmo.Candidates. The system manager uses this information to determine how the device fits into the network topology and thereby establishes communication relationships between a device and its neighbors.

Advertising devices may be discovered using passive scanning, active scanning, or a combination of passive and active scanning.

In passive scanning, the DL periodically listens for advertisements on a series of channels. Generally, a battery-powered passive scanning device will listen frequently when first powered on. If a DL subnet is not discovered quickly, the device may preserve battery life by scanning less frequently, on fewer channels, and/or for shorter periods of time. This can result in substantial delays in subnet joining and/or subnet formation.

Active scanning overcomes some disadvantages of passive scanning. Devices that are configured for active scanning will search for a subnet by periodically transmitting solicitations, which trigger advertisements from neighboring routers in response. The device transmitting the solicitation is called an active scanning interrogator, and the responding device is called an active scanning host. Active scanning hosts expend energy operating their radio receivers while listening for solicitations. Some active scanning hosts have energy available for continuous receiver operation. Active scanning hosts with more limited energy sources may be configured to listen continuously for certain periods of time, such as during network formation.

Link schedules used for joining are not necessarily related to superframe schedules used for normal network operation. Thus, little information about subnet operation needs to be conveyed in the advertisements.

#### **9.1.13.2 Auxiliary sub-header and advertisements**

DL advertisements and solicitations are conveyed in the DL's auxiliary sub-header (DAUX) within the DL header (DHR). See 9.3.1 for an overview of the DHR.

The DAUX is usually absent from a DHR, but shall be included in any DHR if so configured for a particular DL link. A DPDU containing a DAUX may also carry a higher layer payload that is unrelated to the neighbor discovery function.

Transmission of an advertisement is triggered by an advertisement flag as configured within a DL link definition. The advertisement flag indicates that an advertisement shall be transmitted in a superframe's timeslot, in the absence of a higher priority link.

The same DL link definition may also include a DPDU transmission flag, in which case the DL shall check the message queue for matching outbound DPDUs. Thus, the DPDU may simultaneously carry an advertisement and a DSDU payload that is entirely unrelated to the advertisement.

The data capacity of the DSDU is reduced when an advertisement is embedded within the DHR. Some DPDUs on the message queue, particularly messages that have been fragmented at the network layer, may be too long to be combined with an advertisement. Such messages are not candidates for links that are shared with an advertisement, effectively giving the advertisement priority access to those timeslots.

**NOTE** When messages are fragmented by the network layer, the fragment length is set by the network layer before being passed to the DL, with fragment length being configured by the system manager. In configurations where advertisements are infrequently combined with transmit links, the fragment length may be reasonably configured based on a DL payload capacity without the advertisement.

In general, DPDUs containing an advertisement use a DMIC based on the DL subnet's security key, thereby providing an advertisement that can be trusted by devices after they have joined the subnet. This DMIC cannot be validated prior to joining, because an unjoined device does not yet have the DL subnet security key. Therefore, an unjoined device that is scanning for a DL subnet is permitted to process an advertisement in a DAUX subheader even if it is unable to authenticate the DPDU containing the DAUX.

Without the benefit of a DMIC, an unjoined device receiving an advertisement can still use the IEEE FCS as an integrity check. However, the IEEE FCS alone does not filter MPDUs from other protocols based on IEEE. Therefore this standard provides an additional integrity check, not involving a security key, specifically covering the advertisement subheader, as specified in 9.3.5.2.4.3.

#### **9.1.13.3 Burst advertisements**

A burst advertisement is a series of advertisements, transmitted in quick succession on a single channel, within the bounds of a single timeslot. The advertisements within the burst are identical to each other. A link may be configured to transmit or to receive a burst advertisement. A burst advertisement DPDUs shall not contain any payload in its DSDU.

Generally, the intent is for the burst advertisement to provide an opportunity for a passive scanning receiver to use channel sampling techniques to allow for more power-efficient

passive scanning. When advertisements are transmitted in quick succession, advertisement N can essentially serve as an extended PHY synchronization header for advertisement N+1. Thus, a device that is scanning for the advertisement may check the channel for IEEE Std 802.15.4 radio modulation, which can be a relatively quick operation. Once such modulation is detected, the DL can sensibly keep its receiver active until the current DPDU (presumptively advertisement N) is completed, with the expectation that advertisement N+1 will follow shortly thereafter on the same radio channel and with similar signal characteristics.

The transmission of a burst advertisement is specified by the standard, but the detection and reception of a burst advertisement is left to the device designer. Routers are required to transmit burst advertisements when so configured, and I/O devices are required to be capable of receiving them. For reception of a burst advertisement, an unoptimized implementation may simply enable its receiver until a complete advertisement is received. However, opportunities for device-specific optimizations are provided by the standard in the way that burst advertisements are transmitted. The intention is to enable the passive scanning device to quickly sample the radio channel for IEEE Std 802.15.4 direct sequence modulation, for example, by using the radios clear channel assessment capabilities. Once a valid radio signal is detected, the scanning device can sensibly enable its receiver and efficiently scan for a full advertisement DPDU with a reasonable chance of success.

Within each burst advertisement transmit link, the device shall transmit as many advertisements as possible within the time bounds of the link. The first such advertisement PPDU shall start within 96  $\mu$ s (6 PHY symbols) of the link's transmission window (starting at XmitEarliest), and repeat within the transmission window (ending at XmitLatest) with spacing in the range of 192  $\mu$ s to 384  $\mu$ s between consecutive PPDUs. The length of each advertisement's PPDU, including the SHR and PHR prior to the DPDU, shall not exceed 52 octets. The transmission window, which is based on DPDU start time, shall be at least 6.3 ms in duration.

The DL does not calculate the DMIC for burst advertisements, thereby enabling transmission and the PPDUs in quick succession without delays for security processing. Burst advertisements are intended to facilitate network discovery by devices in the provisioned or factory state. Such devices do not process incoming DMICs, because they lack the necessary security key material (see 9.3.5.2.4.3). Therefore, for burst advertisements only, the DMIC field in the header shall use a DMIC length of 4 octets with a fixed value of 0xffffffff. In contrast, advertisements that are not burst advertisements shall calculate a DMIC using the DL's security key policy. Network participants, with DL security material, shall not process incoming burst advertisements due to DMIC mismatch.

**NOTE 1** The maximum length of a dedicated advertisement is calculated as follows: 6 octets for the SHR + PHR, 7 octets for an MHR, 1 octet for a DHDR, 2 octets for the DMXHR, a maximum of 29 octets for the DAUX (see Table 119), no DROUT, no DADDR, no DSDU, 4 octets for the DMIC, 2 octets for the FCS.

**NOTE 2** The spacing between burst advertisements within a timeslot is based on two factors. Short frame interface spacing (SIFS) in IEEE Std 802.15.4 is 192  $\mu$ s, and therefore spacing between burst advertisements shall not be less than 192  $\mu$ s. Transmitter jitter of  $\pm 96 \mu$ s is allowed in this standard. Combining these, a burst advertisement spacing of  $288 \pm 96 \mu$ s is the shortest timing that preserves SIFS.

#### 9.1.13.4 Active scanning solicitation and response

In active scanning a new device, acting as an active scanning interrogator, periodically solicits advertisements from active scanning hosts that happen to be in radio range. A DL receiving the solicitation can be configured to respond with an advertisement.

Active scanning is intended for configurations in which an advertising device, in its capacity as an active scanning host, is able to operate its receiver more or less continuously in a slow hopping configuration. Active scanning hosts may be continuously powered. Alternatively they may be energy-constrained devices that run their receivers in a slow hopping configuration for limited periods of time, such as during network formation.

Solicitations are encoded in the DAUX sub-header. The DSDU length of a solicitation shall be zero, that is, the DPDU lacks a network layer payload.

A solicitation, when received by an active scanning host, causes that host to transmit an advertisement in the next timeslot if the DL is so configured. A router receiving a solicitation shall respond by transmitting an advertisement in the next full timeslot if, and only if:

- The default receive link for scanning (Table 165) applies in next timeslot and occurs on the same radio channel as the solicitation; and
- The DL's attribute `dlmo.ActScanHostFract` is configured for response to solicitations. `dlmo.ActScanHostFract` indicates the fraction of time that the DL should respond when it receives an active scanning solicitation. The default of 0 indicates that the device is not configured as an active scanning host and that it shall not respond to solicitations. A value of 255 indicates that the device should always respond to solicitations. A value between 1 and 254 indicates that the DL shall calculate a randomized number from 1 to 255 each time it receives a solicitation, and shall not respond with a solicitation if the result greater than `dlmo.ActScanHostFract`; and
- The DL is configured to respond to the subnet ID included in the advertisement; as described in 9.4.2.20. A solicitation may be configured to include a subnet ID to limit respondents to a desired set of subnets.

The next full timeslot, in this context, shall be defined as the next timeslot that starts following the end of the solicitation's PPDU plus 1 ms. Within that next timeslot, the advertisement shall be transmitted using timing as defined in the default transmit template in Table 166, even if that default is overwritten by an alternative configuration.

An active scanning interrogator, which is presumably not synchronized with network timing, should enable its radio to receive an advertisement with a DPDU that starts as early as 3212  $\mu$ s following the end of the solicitation (1 ms plus the 2212  $\mu$ s from Table 166). Following that time, it should keep its receiver enabled long enough to receive an advertisement DPDUs beginning at any time during a full timeslot duration. The active scanning interrogator should use its own timeslot duration as the assumed timeslot duration of the active scanning host. Therefore, when solicitations are used, the active scanning interrogator should be configured with a timeslot duration that matches or exceeds the timeslot duration of the target network.

A solicitation is triggered by a link that is configured as a solicitation.

To support passive scanning by new devices, active scanning hosts may also be configured to transmit advertisements periodically.

It may be necessary to suppress solicitations, to account for situations where it is unsafe or illegal for a device to operate its transmitter without authorization. To address such situations, the DLMO attribute `dlmo.RadioSilence` provides a mechanism to disable solicitations along with all other DL transmissions.

Solicitations are disabled by default. Solicitations are not used in the default configuration, and `dlmo.ActScanHostFract` defaults to zero.

### 9.1.13.5 Continuous scanning

Neighbor discovery should be an ongoing process even after a device has joined the network. Ongoing scans can, over time, help to form a more optimal network. Mains powered routers that spend a substantial portion of their time listening can, over time, receive many advertisements from nearby routers. Battery-powered devices cannot spend a high percentage of their time listening, but even if they just sample the channel periodically, they can, over extended periods of time, build a comprehensive picture of neighboring routers. If the DL subnet is configured with a coordinated schedule of advertisements, such scanning can be performed more efficiently.

A system management function may establish an overall subnet schedule for advertisements, and a joined device may be provisioned with a schedule of receive links to ensure that such advertisements are heard over time. When used, this approach enables devices on the subnet to find neighbors efficiently. Additionally, a low duty cycle device may be configured to use these scheduled advertisements to remain time-synchronized with the network.

Advertisements are authenticated with a DMIC, to enable devices that have joined the subnet to rely on scheduled advertisements as a trusted source of timing and connectivity information.

### **9.1.14 Neighbor discovery and joining – data link layer considerations**

#### **9.1.14.1 General**

The DL provides a configurable mechanism to discover neighboring devices.

During provisioning, the DL is configured to scan for neighbors that can act as proxies in the join process. When an advertisement is received from a candidate neighbor, the DL uses information in the advertisement to create communication links to and from that neighbor, and then provides the neighbor's addressing information to the DMAP. For the remainder of the join process, the DL provides communication support to upper layers by passing DPDUs to and from the neighbor.

After joining the network, the DL is implicitly or explicitly instructed by the system manager to scan for new neighbors for a designated period of time, using superframes and links provided by the system manager. The result of this scan is reported as neighbor information to the system manager that uses the information to provide the device with an optimal configuration within the DL's mesh. The DL then continues accumulate information about new candidate neighbors throughout its life cycle, and this information is periodically reported to the system manager to facilitate configuration of improved and adaptive mesh configurations.

The network joining process from the DL's point of view can be informatively summarized as follows:

- The DL, in its factory state, searches for an advertisement from the provisioning device. This search is built into the device as defined by the standard.
- The DL receives an advertisement from the provisioning device. This advertisement, with subnet ID = 1, provides a compressed but fully functional configuration for the DL's state machine. The DL uses this configuration to communicate with the provisioning device. During provisioning, a different DL configuration, that is subsequently used to search for the target network, is written to the device provisioning object (DPO).
- At the end of provisioning, the provisioning device sets Join\_Command=1, indicating successful completion of the provisioning process and causing the DL to reset to the provisioned state. The DL defines that reset as first resetting the DL to its factory state, thus erasing the material from the provisioning device's advertisement, and then initializing the DL with the settings stored in the DPO.
- The DL then commences operation of its state machine, using the configuration as initialized by the DPO. This configuration from the provisioning device should be matched to the target network to facilitate efficient network discovery. For example, if the target network is configured to transmit advertisements on three channels, the DPO might reasonably configure the DL to scan those same three channels.
- The discovery process is successful when the DL receives an advertisement from the target network. This advertisement contains a compressed but fully functional DL configuration that can be used for communication with the system manager through the router that transmitted the advertisement.
- If the join process times out, the DL is reset to the provisioned state. The configuration derived from the advertisement is erased, the DPO configuration is re-established, and the DL resumes its search for a target network.
- If the join process is successful, the DL's joining configuration persists until explicitly updated by the system manager. Thus, at the end of the join process, the DL typically retains an interim connection with the system manager through the advertising router.
- Following a successful join, the DL searches for a set of candidate routers that can be used for communication. After a configurable period of time, defaulting to 60 seconds, the

DL reports this list of candidates to the system manager. This information is used by the system to replace the interim connection with a more permanent and resilient DL configuration.

#### 9.1.14.2 Data link layer states

When a device is manufactured, the DL is in its default state. Attributes in the default state are defined by the standard.

In the default state, the DL shall periodically scan for a provisioning network, using a search procedure defined by the standard. When the DL receives one or more advertisements from a provisioning device in a provisioning network, the DL shall use information in one of the advertisements to establish a superframe with links that can be used to communicate with the selected provisioning device. The DL then informs the DPO (device provisioning object) that a DL-level connection has been established, and switches the DL to the provisioning state.

In the provisioning state, the DL halts the search procedure defined by the standard. Instead, the DL operates its state machine according to a superframe with links that were provided by the provisioning device's advertisement. During the provisioning process, the DL provides communication services to upper layers by operating its state machine according to the advertised superframe and links, so that provisioning APDUs can be passed between the DPO and the provisioning device through the DL.

If the provisioning process times out, the device puts the DL back into its default state, and the DL resumes its default search procedure for a provisioning network.

If the provisioning process is successful, the DPO provides the DL with a set of attributes, including subnet information, superframes, and links, that the DL can use to search for the target DL subnet(s). The DL then switches from the provisioning state to the provisioned state, and operates its state machine as configured in the superframes and links that were provided by the DPO.

The switch from the provisioning state to the provisioned state is triggered when the provisioning device sets Join\_Command=1 (Table 10). When that occurs, the DL is reset to its provisioned state and then operates its state machine as configured in order to search for a target network.

In general, a DL reset to provisioned state is accomplished in two steps. First, the DL is reset to its default state. Then, information from the DPO's Target\_DL\_Config attribute is applied to the DL. This provides a set of attributes, including subnet information, superframes, and links, that the DL uses to search for the target network and corresponding subnets. See 14.8.

If the device is provisioned through an out-of-band mechanism, the provisioning state may be bypassed and in that case the DL transitions directly from the unprovisioned state to the provisioned state.

The DPO retains a copy of the information that was used to provision the DL, providing a means to reset the DL back to its provisioned state by putting the DL into its default state and then adding the provisioned attributes from the DPO.

A DL in the provisioned state operates its state machine as configured in its provisioned superframes and links. The provisioned superframes and links should be matched to the operating characteristics of the target DL subnet(s), so that a target network is efficiently discovered when the device and a target DL subnet are in proximity to each other. The result is that the DL receives at least one advertisement from at least one proxy device that is participating in a target the DL subnet. The DL uses information in one of these advertisements to establish a superframe with links that can be used to communicate with that proxy device. The DL then informs the DMAP that a DL-level connection has been established to a neighboring proxy, and the DL switches from the provisioned state to the joining state.

When the DL enters the joining state, it retains its configuration from the provisioned state (see 14.8), and adds the superframe, links and other attributes defined or implied by the advertisement. The DL then provides communication services to upper layers during the joining process, by operating its state machine according to the advertised superframe and

links with the result that joining APDUs are passed between the DMAP and proxy device through the DL.

If the DMAP's joining process times out, the device resets the DL back into its provisioned state and the DL resumes scanning for a network using the provisioned superframes and links.

If the DMAP's joining process is successful, the advertised superframes and links continue to be used temporarily for communication with the system manager. The DPO retains the information needed to reset the device back to the provisioned state in the event of a device reset.

When the DL is first placed in its joined state, it has a single connection to the DL subnet through the neighboring proxy device, using the superframe and links defined in the advertisement. This single connection, selected implicitly when the DL selected the proxy device, lacks path diversity, and may be suboptimal for any number of reasons. Therefore, the system manager shall provide instructions for the DL to search for several neighbors and to report the result when the search is completed. Simple instructions may be provided through the same advertisement that established the initial connection to the proxy device, or more elaborate search instructions may be provided by the system manager immediately after the device joins the network. In either case, the DL provides the system manager with a list of candidate neighbors soon after it joins the network, with 60 s being the default reporting time as controlled by the attribute `dlm0.DiscoveryAlert`. The system manager analyzes this list of candidate neighbors and then provides the DL with an updated configuration that includes path diversity (mesh) and generally provides a more optimal DL subnet connection.

To support security processing of each DPDU, the DMAP needs the EUI-64 of the advertising device during both the provisioning and joining process. Since the advertisement provides only the advertising router's 16-bit address, the DL shall acquire the EUI-64 from the neighbor when communication begins. The neighbor's EUI-64 shall be acquired by using a transmit link to interrogate the neighbor using a DPDU without a payload (zero length DSDU), and setting the DHDR request EUI-64 bit (bit 5, Table 118) to a value of 1, causing the neighbor's EUI-64 to be returned in the acknowledgement.

#### **9.1.14.3 Consolidated configuration information to the data link layer**

The DPO maintains the attribute `Target_DL_Config` that includes the settings for various attributes in the DL. This OctetString is provided to the DL at the end of the provisioning process, and is retained by the DPO in the event that it needs to reset the DL back to its provisioned state.

The `DL_Config_Info` OctetString contains a collection of attributes that the DPO provides to the DL in order to establish the provisioned state. Each attribute is expressed as a tuple including the attribute number, followed by an OctetString that contains the new attribute value. The structure of `DL_Config_Info` is shown in Table 103.

**Table 103 – DL\_Config\_Info structure**

octets	bits
	7      6      5      4      3      2      1      0
1 octet	N (Number of attributes)
1 octet	AttributeNumber <sub>1</sub> (Unsigned8)
OctetString Length	NewAttribute <sub>1</sub> (OctetString)
.....	
1 octet	AttributeNumber <sub>N</sub> (Unsigned8)
OctetString Length	NewAttribute <sub>N</sub> (OctetString)

Several of the attributes in `DL_Config_Info` are indexed OctetStrings. In these cases, `NewAttributex` is a new row entry. By DL convention, each row entry in an indexed OctetString attribute includes the row index as its first field.

DL\_Config\_Info can be used to configure any read/write attribute in the DL. At a minimum, DL\_Config\_Info shall configure:

- AdvFilter, which provides a filter so that the DL can select superframes that are of interest.
- At least one superframe, and at least one link, that can be used by the DL in searching for advertisements.

Timeslot templates used for searching for the DL subnet, if different from the default timeslot templates, shall be provided to the field device during the provisioning process.

DL\_Config\_Info shall not be used except through the DPO.

The DL's provisioned attributes shall be retained by the DPO so that the DL can be reset to its provisioned state.

Superframe operation may be delayed or disabled by setting the IdleTimer field within the superframe. For example, two superframes may be provisioned in a device, with superframe #1 searching aggressively for the DL subnet and superframe #2 searching on a low duty cycle. The IdleTimer of superframe #1 might cause that superframe to time out a few minutes after the DL is configured. Alternatively, a superframe might be configured with an IdleTimer so that it is idle until some future time. If the DL is reset to the provisioned state, the superframe idle timers shall be reset to the originally provisioned state as well.

A device in the provisioned state shall operate its DL clock and increment TAI time. This enables the device to operate its superframes as provisioned to discover candidate subnets. During the join process, a revised TAI time will be received in advertisements and the DL clock reset accordingly.

The device might completely lose its time sense in the provisioned state, such as due to removal of a battery. Complete loss of time sense in a provisioned device shall trigger a reset of the DL to its provisioned state.

#### **9.1.14.4 Scanning for neighbors in the unprovisioned state**

An unprovisioned DL begins in the system default state. Its DL configuration includes the five default channel hopping patterns and the three default timeslot templates. Its superframes, links, graphs, and routes are blank.

Before attempting to join the plant network, the unprovisioned device needs to establish contact with a provisioning device and be transitioned to the provisioned state. This may occur through an out of band mechanism, such as a wired modem or an infrared link. Additionally, as will be described here, a provisioning device can activate the DL of an unprovisioned device by transmitting burst advertisements.

The standard defines an unprovisioned DL's search procedure for a provisioning networks advertisement. The search procedure is radio silent, not involving solicitations or any other transmission until an advertisement is received from a provisioning network.

An unprovisioned DL shall scan for a provisioning network's advertisements on channels 4 and 14, corresponding to IEEE Std 802.15.4 channels 15 and 25, if radio regulations so permit (see Figure 59). In each of these channels, the unprovisioned device shall scan for a burst advertisement (see 9.1.13.3) at a fixed interval of exactly 0.25 s, 0.5 s, 1 s, 2 s, 4 s, 8 s, 16 s, or 32 s. When the device is powered on or physically reset, it shall scan at the shortest interval of 0.25 s for at least 10 s, and then may gradually decrease the interval as necessary to preserve its energy source.

SubnetID=1 is reserved for the provisioning network. Therefore, only advertisements with SubnetID=1 shall be considered by a DL in the unprovisioned state.

#### **9.1.14.5 Scanning for neighbors in the provisioned state**

Once a device is in the provisioned state, it shall use its provisioned superframes and links to scan for a DL subnet of interest. The device may discover multiple advertisement routers and then select one to be used as a proxy in the join process.

### 9.1.14.6 Scanning for neighbors after joining the network

A device joins the network through a single neighbor that is sending advertisements. Initial contracts are established based on a route that is not necessarily optimal, through the joining proxy device. Having established a single connection through the join process, the DL shall be configured by the system manager to immediately search for additional and alternative neighbors in order to support a more optimized mesh configuration.

A DL in the joined state can be configured by the system manager to passively scan for advertisements; by configuring the DL with links and superframes that enable the DL radio receiver at scheduled times. The device may be configured to actively scan for advertisements using solicitations.

The NeighborDiscovery alert (see 9.6.2 and 9.4.2.24) provides a mechanism for the DL to push this information to the system manager. The alert is intended for the following scenarios:

- Within 15 seconds of entering the joined state, the DL shall be configured with superframes and links that search for advertisements from routers that are in range, and select the most promising candidates. The configuration can be accomplished through the advertisement itself, prior to joining, as described in 9.3.5.2.4.2. Alternatively, the configuration can be provided by configuring DLMO attributes after joining. After a configurable elapsed amount of time, as defined by the attribute dlmo.DiscoveryAlert, the DL shall use the neighbor discovery alert to send the list of candidates to the system manager. A superframe dedicated to this initial search can be configured to automatically time out through its IdleTimer field.
- The device should be configured by the system manager to continuously passively and/or actively scan for advertisements on an ongoing basis. Over time, this enables the device to build a list of candidate neighbors. The HRCO may be configured to periodically transmit this candidate neighbor table, along with neighbor diagnostics, to the system manager. The system manager may alternatively read the candidate neighbor table, dlmo.Candidates, on its own schedule.
- A device shall also use the neighbor discovery alert whenever a connectivity issue is reported through the DL\_Connectivity alert (see 9.6.1). This provides an up-to-date picture of neighborhood connectivity, enabling the system manager immediately to consider alternative solutions to the reported DL connectivity problem.

## 9.1.15 Radio link control and quality measurement

### 9.1.15.1 General

Signal quality information is accumulated in the DL and reported through the DMAP. In support of these higher-level functions, the DL provides primitives that report signal quality information. The DL also provides attributes that enable the system manager to control radio emissions.

### 9.1.15.2 Performance metrics

Numerous performance metrics can be accumulated by the DL on a per-neighbor basis, configured by the system manager and reported through the DMAP. (See 9.4.3.9.)

The DL can be configured by the system manager to accumulate the following types of performance data on a per-neighbor basis:

- Received signal strength indicator (RSSI) and received signal quality indicator (RSQI). This standard defines practices for RSSI and RSQI reporting to facilitate consistency, so that signal characteristics may be compared and appropriate links selected in multi-vendor situations.
- For transmissions: counts of successful transmissions, CCA backoffs, unicast errors, and NACKs.
- Count of received DPDUs, excluding ACK/NACKs.
- Diagnostics of clock corrections.

Per-channel diagnostics are also collected and consolidated for all neighbors.

RSSI shall be reported as a signed 8-bit integer, reflecting an estimate of received signal strength in dBm. RSSI reports shall be biased by +64 dBm to give an effective range of -192 dBm to +63 dBm. For example, a reported RSSI value of -16 corresponds to a received signal strength of -80 dBm. It is recognized that available integrated radios might not be capable of providing these physical units directly or at high accuracy. Nonetheless, device designers should approximately map available diagnostics to a reasonable estimate of RSSI units so that system managers have a consistent basis for making routing decisions among devices.

RSQI shall be reported as a qualitative assessment of signal quality, with higher number indicating a better signal. A value of 1-63 indicates a poor signal, 64-127 a fair signal, 128-191 a good signal, and 192-255 an excellent signal. A value of zero indicates that the chipset does not support any signal quality diagnostics other than RSSI.

RSSI is a quantitative measurement, mapped to physical units. RSQI is a qualitative measurement.

RSQI metrics are intended to be particularly useful when comparing different entries in dlmo.Candidates. A DL implementation may use innovative techniques to report comparisons of the likely link quality of different candidate neighbors. RSQI entries in dlmo.Candidates reported from a given device may be reasonably compared to each other, and fine distinctions can be taken as meaningful. For example, differences within the range of good signals can be reasonably taken into consideration if the RSQI metrics are from the same device. RSQI may also be compared across different devices, but fine distinctions are unlikely to be as meaningful. For example, the distinction between a fair and an excellent link is likely to be meaningful even if reported from unlike devices, but distinctions between different levels of good links has no standard meaning if reported from different devices.

#### **9.1.15.3 Accumulating and reporting diagnostic information**

The system manager establishes a DL communication relationship between a device and its neighbor by adding an entry to the device's dlmo.Neighbor attribute. Each such entry specifies a level of diagnostics to be collected, through the field dlmo.Neighbor[].DiagLevel. For each neighbor, diagnostics may be collected at a baseline level, or a detailed level including clock diagnostics.

Per-channel diagnostics are accumulated and consolidated for all neighbors, in the attribute dlmo.ChannelDiag.

When the dlmo.Neighbor[].DiagLevel field is set for a particular neighbor, the DL shall create corresponding entries in the read-only attribute dlmo.NeighborDiag. NeighborDiag values are accumulated from the time that the dlmo.NeighborDiag entry is created.

Three mechanisms are provided for reporting diagnostic information contained in dlmo.NeighborDiag and dlmo.ChannelDiag:

- The health reports concentrator object (HRCO), described in 6.2.7.7, can be configured to report any attribute in the DL on a periodic basis. dlmo.NeighborDiag entries and dlmo.ChannelDiag can be reported through that mechanism.
- Diagnostic information can be retrieved at any time by the system manager, by reading the applicable attributes.
- Diagnostic information can be reported by the DL on an exception basis, through the DL\_Connectivity alert.

Diagnostics include a combination of levels, such as RSSI, and counters, such as a count of acknowledgements.

Levels are accumulated as exponential moving averages (EMA). The level is initialized with the first data value. Thenceforth, each new data value is accumulated into the EMA level as follows:

$$\text{WHERE EmaLevel}_{\text{NEW}} = \text{EmaLevel}_{\text{OLD}} + \alpha * (\text{NewData} - \text{EmaLevel}_{\text{OLD}})$$

The smoothing factor  $\alpha$  is expressed as an integer percentage in the range of 0 to 100, and is configured by the system manager through the attribute `dlmo.SmoothFactors`. See 9.4.2.25.

Counters in `dlmo.NeighborDiag` are accumulated as `ExtDUIint` unsigned integers. When a counter reaches its maximum value of 32767, it shall report the maximum value. Counters shall be reset to zero whenever the row is reported through the HRCO or retrieved through a read operation. Reporting the value through the `DL_Connectivity` alert shall not reset any counters.

#### 9.1.15.4 Radio silence

The DL can be configured to transmit only when actively participating in a DL subnet. This behavior is configured by the `dlmo.RadioSilence` attribute, which designates a timeout period for DL subnet participation, in seconds. For example, if the `dlmo.RadioSilence` attribute is set to the default of 600 s (10 minutes), the DL silences its radio transmitter 10 minutes after losing communication with the DL subnet. When all devices on a DL subnet are configured for radio silence, it is possible to disable the subnet entirely, even if some devices do not receive an explicit command to disable communications.

When a valid time update is accepted by the DL from an advertisement or an acknowledgement, the DL internally records the current time as the radio silence time reference. If the DL does not accept another time update in the subsequent time period designated by `dlmo.RadioSilence`, the DL shall become silent by ignoring all of its configured transmit links, including solicitations. In the radio silent state, the DL continues to operate its radio receiver as per its scheduled receive links, but without transmitting acknowledgements in the absence of a clock update.

For example, suppose the DL receives a time update at 01h:02m:03s, and `dlmo.RadioSilence` is set to 600 s (10 minutes). If the DL does not receive another time update by 01h:12m:03s, i.e., 10 minutes later, it will silence its radio at that time.

If `dlmo.RadioSilence` is configured as zero, the feature is disabled.

Radio silence is the default. The default subnet discovery procedure does not use solicitations, and `dlmo.RadioSilence` defaults to 600 s.

Support of the `dlmo.RadioSilence` attribute is required for all devices.

The radio silence profile limits the permitted range of the `dlmo.RadioSilence` attribute. The radio silence profile is reported to the system manager on joining through `dlmo.DeviceCapability`. A DL with the radio silence profile shall reject updates to `dlmo.RadioSilence` that are outside of the range from 1 to 600 s, thus ensuring that such a device will never transmit a DPDU once it has lost contact with the subnet for 10 minutes.

Temporary radio silence can be accomplished with another attribute `dlmo.RadioSleep`. When `dlmo.RadioSleep` is set to a positive value, the device treats all links, including receive links, as idle for the designated number of seconds. Activation of `dlmo.RadioSleep` shall be slightly delayed to allow for transmitting an application layer acknowledgement for the DMAP TPDU that causes the attribute to be set. When the sleep period is over, `dlmo.RadioSleep` is automatically reset to zero, indicating that the feature is disabled.

#### 9.1.15.5 Radio transmit power

The standard provides the system manager with a degree of control over radio transmit power, through the attribute `dlmo.RadioTransmitPower`.

`dlmo.RadioTransmitPower` is used to control the DL's radio transmit power level, in dBm EIRP. It defaults to the device's maximum supported power level. This default is also reported to the system manager during the join process through `dlmo.DeviceCapability`.

When `dlmo.RadioTransmitPower` is changed by the system manager the device should not transmit at an output power level in excess of `dlmo.RadioTransmitPower`.

In addition, a device may autonomously calibrate its output power level to the minimum level needed to maintain reliable connectivity. To enable this, the DL supports the echoing of signal quality information in acknowledgements, so that an implementation can calibrate the received signal quality at various power levels. See 9.3.5.5.

### 9.1.15.6 Regulatory environment

#### 9.1.15.6.1 Country code

The provisioning device and/or the system manager can inform the DL of regulatory considerations through the attribute `dlmo.CountryCode`, a 16-bit unsigned integer where:

- Bits 0-9 provide a 10-bit country code, using ISO 3166-1 numeric three-digit country codes.
- Bit 10 indicates whether ETSI rules apply (0=no, 1=yes). A device should operate in compliance with ETSI rules when Bit10=1.
- Bit 11 indicates whether FCC rules apply (0=no, 1=yes). A device should operate in compliance with FCC rules when Bit11=1.
- Bit 12 indicates whether a 10 dBm EIRP limit applies (0=no, 1=yes). A device should limit its emissions to 10 dBm EIRP when Bit12=1.
- Bits 13-15 are reserved and shall be set to 0.

The default for `dlmo.CountryCode` is 0x0400, indicating that a device in the default state should comply with ETSI rules.

#### 9.1.15.6.2 Regulatory considerations (INFORMATIVE)

Radio regulations may require pre-configured devices that operate at reduced power levels, depending on where the device is deployed. The identified thresholds (E.I.R.P.) are: 3 mW (Japan), 10 dBm (China), 10 dBm/MHz (ETSI), 20 dBm (ETSI, FH), and 36 dBm (FCC, with a 6 dBi antenna gain restriction).

In some countries, such as France, emission levels on certain channels need to be attenuated. In other countries, such as Korea, the number and range of channels need to be constrained.

ETSI requires frequency hopping across 15 channels for emissions above 10 dBm/MHz, and 20 channels for adaptive frequency hopping. Therefore, spectrum management and selective channel utilization may be constrained in ETSI-compliant installations.

### 9.1.16 Data link layer roles and options

The DL within this standard is designed with the general goal of constraining the range of build options for a field device, while enabling flexible and innovative system solutions.

The DL framework does not require that all DLs are equivalent. For example, some routers, designed as dedicated infrastructure devices, might have a continuous source of energy, powerful processors, and essentially unlimited memory capacity. In contrast, some field instruments may have low-capacity batteries and may lack routing capability.

These distinctions among DL implementations are covered in three general ways in this standard:

- Memory capacity;
- DL capabilities; and
- DL roles.

Every device has a limited amount of memory that is available for DL operations, and the system manager needs knowledge of these limitations in order to configure the device and balance network operation. DL memory is not reported as a single block, but rather as specific capacities of memory for specific purposes. For example, each indexed `OctetString` attribute

supports a limited number of entries, with the capacity available to the system manager as metadata. Similarly, buffer capacity for DPDU forwarding is reported by the device on startup.

Certain DL capabilities are also reported on startup. For example, the device reports the stability of its own clock, as well as a list of radio channels that it can support legally. DL capabilities reported with the join request are enumerated in 9.4.2.23.

DL roles describe the general capabilities of a given DL configuration. For example, a DL implementation may be capable of routing or not. Distinctions of this type have various implications throughout the DL implementation, in terms of minimum memory capacity, device capabilities, and support of various features. The device simply reports which DL roles it supports, and the system manager is then responsible for mapping this into a portfolio of device capabilities. Standard mappings between roles and minimum capabilities are provided in Annex B.

### 9.1.17 Data link layer energy considerations

Devices have different levels of available energy. One device may have a continuous energy source. Another device may have a large battery, but need most of that energy capacity for running a sensor. Yet another device may use energy scavenging as its primary energy source. Different battery chemistries have different characteristics, a given battery chemistry may provide different performance depending on the supplier, and a battery's capacity may vary depending on environmental factors. New battery technologies are likely to emerge with currently unknown performance characteristics. One application might need a 20-year battery life, while a different application might tolerate a 6-month life.

The DL may be configured to be the system manager to consume different amounts of energy. The DL consumes energy in two general ways:

- The DL consumes energy providing wireless service to its own applications. When a device establishes a contract to transmit data every 5 s, the DL consumes a corresponding amount of energy.
- The DL consumes energy acting as a router on behalf of neighboring devices. A device may be configured to transmit advertisements every 10 s. A device may be configured to operate its receiver almost continuously, listening for solicitations. The DL subnet may be configured so that a device forwards 100 DSDUs per minute. All of these scenarios consume energy.

The DL reports a general sense of its capacity to support DL routing operations in certain fields of the `dlmo.EnergyDesign` attribute. This attribute is reported through the `dlmo.DeviceCapability` attribute.

`dlmo.EnergyDesign` indicates the device's designed energy capacity to handle DL operations. This attribute is constant over the life of the device and reflects the device's design, not its current state. A system manager should configure a device within these stated energy limitations:

- `EnergyLife` indicates the device's energy life by design. A positive value provides energy life in days; a negative value provides energy life magnitude in hours. A value of 0x7FFF indicates a continuous power source and no practical DL energy limitations. Other `EnergyDesign` fields describing DL energy capacity are based on this target energy life. Configuration of the DL beyond these stated energy capacities will likely reduce the device's energy life.
- `ListenRate` indicates the DL's energy capacity on average, in seconds per hour, to operate its radio's receiver. ListenRate includes time to receive DPDUs for the device's own application contracts, plus DPDUs being forwarded by the DL on behalf of other devices.
- `TransmitRate` indicates the DL's energy capacity, in DPDUs per minute, to transmit DPDUs on its own behalf and to forward DPDUs on behalf of its neighbors.
- `AdvRate` indicates the DL's energy capacity, in DPDUs per minute, to transmit dedicated advertisement (or solicitation) DPDUs.

EnergyDesign is a constant, and does not reflect the changing state of a device's energy source. The dlmo.EnergyLeft attribute is a dynamic read-only attribute that can be used to report the device's remaining energy capacity. A positive value indicates the remaining life in days, and a negative value indicates the magnitude of the remaining life in hours. A value of 0x7FFF indicates that the feature is not supported. dlmo.EnergyLeft is reported on startup through dlmo.DeviceCapability, and may also be reported periodically through the HRCO.

## 9.2 Data link layer data service access point

### 9.2.1 General

The DDSAP supports the multi-hop conveyance of network payload (DSDU) between devices in a DL subnet.

DD-DATA.request takes a DSDU from the network layer, prepends a DPDU header, and adds it to the message queue. DD-DATA.confirm subsequently reports whether the DSDU was successfully conveyed to a neighboring device in the DL subnet.

DD-DATA.indication indicates the arrival of an incoming DPDU that has reached its final destination within the DL subnet, and passes its DSDU to the network layer.

### 9.2.2 DD-DATA.request

DD-DATA.request is a virtual primitive that accepts DSDU from the network layer, selects the route through the DL subnet, and places a corresponding DPDU on the device's DL message queue.

The semantics of the DD-DATA.request primitive are as follows:

```
DD-DATA.request
  (
    SrcAddr,
    DestAddr,
    Priority,
    DE,
    ECN,
    LH,
    ContractID,
    DSDULength,
    DSDU,
    DSDUHandle
  )
```

Table 104 describes the parameters for DD-DATA.request.

**Table 104 – DD-DATA.request parameters**

Parameter name	Parameter type
SrcAddr (NL source address)	Type: Unsigned16 or Unsigned64
DestAddr (NL destination address)	Type: Unsigned16 or Unsigned64
Priority (priority of the payload)	Type: Unsigned4
DE (discard eligible)	Type: Unsigned1
ECN (explicit congestion notification)	Type: Unsigned2
LH (last hop, NL)	Type: Unsigned1
ContractID (ContractID of the payload)	Type: Unsigned16 or null
DSDULength (payload length)	Type: Unsigned8
DSDU (number of octets as per DSDULength)	Type: Octets
DSDUHandle (uniquely identifies each invocation of this primitive)	Type: Abstract

DD-DATA.request parameters include:

- SrcAddr is the source address of the NSDU. It is normally the 16-bit alias of the NSDU's 128-bit source address, except when it is the EUI-64 of an unjoined device. Subnet ID is implicit, based on dlmo.SubnetID.
- DestAddr is the destination address of the NSDU. It is normally the 16-bit alias of the NSDU's 128-bit destination address, except when it is the EUI-64 of an unjoined device. Subnet ID is implicit, based on dlmo.SubnetID.

- Priority is copied to the DROUT sub-header and indicates the DPDU's priority in DL message queues.
- DE is copied to the DADDR sub-header. DE=1 indicates that the DSDU is eligible to be discarded from a message queue in favor of an incoming DPDU with DE=0, and of equal or higher priority.
- ECN is copied to the DADDR sub-header. See 9.1.9.4.5 for a discussion of ECN.
- LH is copied to the DADDR sub-header. A value of 1 indicates that the DSDU entered the DL subnet through a backbone router, and therefore shall not exit the DL through a backbone router to avoid circular routes at the network layer. This enables the network layer to elide the IPv6 hop limit field. Logically, LH is carried by the DL on behalf of the NL, and LH shall not be changed by the DL.
- ContractID may be used by the DL in route selection.
- DSDULength indicates the number of octets contained in the DSDU (DPDU payload).
- DSDU is the set of octets forming the payload. It may be implemented as a pointer to memory that is shared among layers.
- DSDUHandle is an abstraction that connects each invocation of DD-DATA.request with the subsequent callback by DD-DATA.confirm.

### 9.2.3 DD-DATA.confirm

DD-DATA.confirm is a virtual primitive that reports the results of a request to transmit network payload (DSDU) that was previously placed on the DL message queue by DD-DATA.request.

Table 105 describes the parameters for DD-DATA.confirm.

**Table 105 – DD-DATA.confirm parameters**

Parameter name	Parameter type
DSDUHandle (identifier for the payload)	Type: Abstract
Status (see Table 106)	Type: Enumeration

Table 106 specifies the value set for the status parameter.

**Table 106 – Value set for status parameter**

Value	Description
SUCCESS	Operation was successful.
FAILURE	Operation was unsuccessful; operation timed out.

NOTE Error handling between the DL and network layers is an internal device matter, not visible across any wireless interfaces, and therefore is not standardized.

### 9.2.4 DD-DATA.indication

DD-DATA.indication is a virtual primitive that indicates the receipt of a network-layer payload (DSDU). A DPDU does not trigger a data indication until reaches its destination on the DL subnet.

The semantics of the DD-DATA.indication primitive are as follows:

```
DD-DATA.indication
(
  SrcAddr,
  DestAddr,
  Priority,
  DE,
  ECN,
  LH,
  DSDULength,
  DSDU
)
```

Table 107 describes the parameters for DD-DATA.indication.

**Table 107 – DD-DATA.indication parameters**

Parameter name	Parameter type
SrcAddr	Type: Unsigned16 or Unsigned64
DestAddr	Type: Unsigned16 or Unsigned64
Priority (priority of the payload)	Type: Unsigned4
DE (discard eligible)	Type: Unsigned1
ECN (explicit congestion notification)	Type: Unsigned2
LH (last hop, network layer)	Type: Unsigned1
DSDULength (payload length)	Type: Unsigned8
DSDU (number of octets as per DSDULength)	Type: Octets

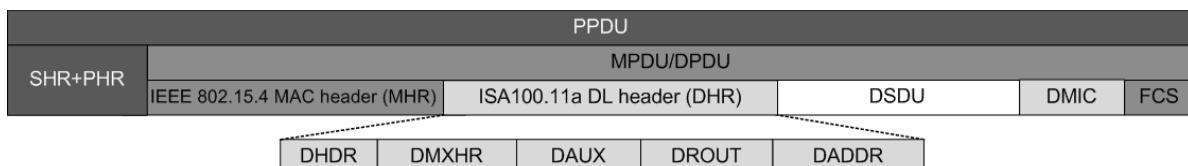
DD-DATA.indication parameters include:

- SrcAddr is the source address of the NSDU. It is normally the 16-bit alias of the NSDUs 128-bit source address, except when it is the EUI-64 of an unjoined device. Subnet ID is implicit, based on dlmo.SubnetID.
- DestAddr is the destination address of the NSDU. It is normally the 16-bit alias of the NSDUs 128-bit destination address, except when it is the EUI-64 of an unjoined device. Subnet ID is implicit, based on dlmo.SubnetID.
- Priority is included in the DROUT sub-header and may be used by the network layer for subsequent routing. ContractID, if required by the network layer, is not carried within the DL header.
- DE provides the value of the DE bit copied from the incoming DADDR sub-header.
- ECN provides the value of the ECN bit copied from the incoming DADDR sub-header, and corresponds to the ECN bit described in IETF RFC 3168. See 9.1.9.4.5 for a discussion of ECN.
- LH provides the value of the LH bit copied from the incoming DADDR sub-header.
- DSDULength indicates the number of octets contained in the DSDU (DPDU payload).
- DSDU is the set of octets forming the payload. It may be implemented as a pointer to memory that is shared among layers.

### 9.3 Data frames and acknowledgements

#### 9.3.1 General

The structure of the standard DL protocol data unit (DPDU) is shown in Figure 88.



**Figure 88 – PPDU and DPDUs structure**

The DL header reflects the DL stack structure described in 9.1.4, including:

- MAC header (MHR): This is the MAC header exactly as specified by IEEE Std 802.15.4. The FCS at the end of the DPDU is logically part of the MHR.
- DL header (DHR): DL header information follows the MHR. Sub-headers within the DHR include:
- DHR header (DHDR): DHDR includes settings for various DL selections and a version number.
- DHR MAC extension sub-header (DMXHR): Additional fields, not specified by IEEE Std 802.15.4, are needed to send a DPDU to an immediate neighbor and to receive an immediate acknowledgement. The DMIC following the DSDU is logically part of the DMXHR.
- DHR auxiliary sub-header (DAUX): Some DPDUs include auxiliary information to facilitate neighbor discovery, time propagation, information exchange, and command exchange among immediate neighbors. The DAUX sub-header is frequently absent. DAUX shall be included in dedicated advertisement or solicitation DPDUs, or alternatively it may be embedded in unrelated DPDUs.
- DHR routing sub-header (DROUT): DROUT contains information needed to route the DPDU through the DL subnet. DROUT shall include DPDU priority class and forwarding limit, plus either GraphID or source routing.
- DHR address sub-header (DADDR): DADDR contains the NSDUs source and destination address, along with upper layer fields (ECN, DE, and LH) that are carried by and visible to the DL.

The DL service data unit (DSDU) is the DL payload, and is also the NPDU as defined in Clause 10.

In certain DPDUs generated internally by the DL, such as dedicated advertisements and solicitations, there is no DROUT, no DADDR, and no upper layer payload, and the DSDU has a length of zero.

### **9.3.2 Octet and bit ordering**

#### **9.3.2.1 General**

Except in the DL, this standard uses most significant octet (MSB or big-endian) transmission and documentation conventions, following the precedent set by ISO/IEC, IETF, and many others. That is:

- For multi-octet values, the most significant octet is transmitted first.
- Octet documentation shows bit 7 on the left and bit 0 on the right.

However, IEEE Std 802.15.4 uses least significant octet (LSB or little-endian) conventions. That is:

- For multi-octet values, the least significant octet is transmitted first.
- Octet documentation shows bit 0 on the left and bit 7 on the right.

Bit transmission order within an octet is handled at the PhL. The MSB and LSB discussion in the DL is limited to the ordering of octets.

The IEEE specification is not entirely consistent on this point. IEEE Std 802.15.4 security sub-headers use MSB transmission and documentation conventions.

As a result, the DL is unavoidably mixed-endian, with some sections using MSB and others using LSB.

Generally, the standard DL headers follow IEEE Std 802.15.4 conventions, as follows:

- This standard, except for DL and MAC headers, follows MSB conventions.
- Standard DL and MAC headers, follow LSB conventions, with some clearly indicated exceptions in the DL header.

- Within the DL, security sub-headers follow MSB conventions, following the IEEE Std 802.15.4 precedent.
- All fields within the DL header are documented showing bit 7 on the left and bit 0 on the right, following the convention of this standard.
- DLMO attributes, accessible to the system manager through the DMAP, are AL information, and as such generally use MSB conventions. Some exceptions are made for fields that interact directly with LSB headers.

Octet ordering is explicitly noted in various parts of the DL specification. By convention, octet 0 is the least significant octet. LSB indicates that the least significant octet (octet 0) is transmitted first, and the most significant octet (octet n) is transmitted last. MSB indicates the reverse.

### 9.3.2.2 Extensible data link layer unsigned integers

The DL specification uses a construct called ExtDLUInt for compressed transmission of unsigned integers. This is not a standard type, and as such ExtDLUInt only appears in DL headers and within DL defined octet strings. It is used to indicate compressed encoding of a 15-bit unsigned integer, and is not used in conveying elemental data. Since this is not used for conveying elemental data, it is not specified as a standard application layer supported data type.

An ExtDLUInt shall be transmitted as one octet when its value is in the range of 0-127, and as two octets when its value is in the range of 128-32767, with encoding as shown in Table 108 and Table 109. Bit 0 in the first octet indicates whether one or two octets are transmitted. Octet ordering is always as shown here, with the length indicated in the bit 0 of the first transmitted octet.

**Table 108 – ExtDLUInt, one-octet variant**

octets	bits								
	7	6	5	4	3	2	1	0	
1	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$		Selection=0

**Table 109 – ExtDLUInt, two-octet variant**

octets	bits								
	7	6	5	4	3	2	1	0	
1	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$		Selection=1
2	$2^{14}$	$2^{13}$	$2^{12}$	$2^{11}$	$2^{10}$	$2^9$	$2^8$	$2^7$	

### 9.3.3 Media access control headers

#### 9.3.3.1 General

This standard uses a MAC header format that is compliant with IEEE Std 802.15.4, followed by extensions that are particular to this standard. Only IEEE data frames are used by this standard.

This standard does not use IEEE Std 802.15.4 security. Instead, similar security is handled in the DMXHR. DL security in this standard is similar to IEEE Std 802.15.4 security. The main difference is that this standard incorporates the device's shared sense of time in the nonce, resulting in more compact messages and improved resistance to replay attacks.

#### 9.3.3.2 Media access control header

The format of the standard MHR is specified in IEEE Std 802.15.4 and is summarized in Table 110.

**Table 110 – Data frame MHR**

octets	bits							
	7	6	5	4	3	2	1	0
2 octets	Frame control (LSB)							
1 octet	Sequence number							
0 or 2 octets	PAN ID (LSB)							
0,2 or ,8 octets	Destination address (LSB)							
0, 2, or 8 octets	Source address (LSB)							

The length of the MHR is usually 9 octets, including a PAN ID and 2 16-bit addresses, with these exceptions:

- Solicitations do not use MAC addressing, so the MHR is 3 octets in length.
- Dedicated advertisements do not use destination addresses, so the MHR is 7 octets in length.
- DPDUs to or from an unjoined device have one 16-bit address and one 64-bit address, so an MHR addressed to or from unjoined device is 15 octets in length.

The default DL payload capacity, dlmo.MaxDsduSize, is based on an MHR length of 15 octets, so that it can be used to make fragmentation decisions for unjoined devices.

As shown in Table 110, fields include:

- Frame control. For this field, subfields are as follows:
  - Frame type shall be data.
  - Security shall be disabled, as it is handled in the DMXHR.
  - Frame pending shall be false.
  - Ack.Request shall be false. The IEEE Std 802.15.4 ACK is not used by this standard.
  - When both a destination address and a source address are included, PAN ID compression shall equal one to indicate that the same PAN ID is used for both addresses. Otherwise, it shall be zero.
  - Addressing modes are typically 16-bit, with exceptions as described below. 64-bit addressing is used by the DL prior to being joined, and destination addresses shall be absent in dedicated advertisements and solicitations.
  - Frame version shall be 0x01.
  - Sequence number. Used by the security sub-layer, as described in 7.3.2.4.10. IEEE Std 802.15.4 requires that the sequence number is incremented after each use for each instance of the DL, so that the sequence number is unique for all DPDUs and ACK/NACKs.
  - PAN ID shall match dlmo.SubnetID and shall be absent in solicitations. If there is a source and destination address, this is a destination PAN ID, with the source PAN ID inferred to be the same. If there is no destination address, such as in an advertisement, it is a source PAN ID. In a solicitation, where there is neither source nor destination address, this field is elided. See 9.1.10.2.
  - Destination address is normally a 16-bit alias for a 128-bit network layer address. An EUI-64 shall be used to address devices that have not yet received a 16-bit alias address. The destination address shall be absent in dedicated advertisements and solicitations.
  - Source address is normally a 16-bit alias for a 128-bit network layer address. An EUI-64 shall be used to identify devices that have not yet received a 16-bit alias address. The source address shall be absent in solicitations.

### 9.3.3.3 Data link layer header sub-header

The structure of the DHDR is shown in Table 111.

**Table 111 – DHDR frame control octet**

octets	bits							
	7	6	5	4	3	2	1	0
1 octet	ACK needed 0=No ACK/NACK 1=ACK/NACK expected	Signal quality in ACK 0=no 1=yes	Request EUI-64 0=no 1=yes	Include DAUX 0=no 1=yes	Include slow hopping offset 0=no 1=yes	Clock recipient 0= not DL clock recipient 1= DL clock recipient	DL version Always 01	

The DHR is always 1 octet in length.

As shown in Table 111:

- Bit 7 shall indicate whether an ACK/NACK is required by a unicast recipient. (Bit 7 is meaningful only for a unicast DPDU.)
- Bit 6 shall indicate whether the receiving device should report signal quality information in its ACK/NACK. (Bit 6 is meaningful only if Bit 7 is 1, indicating that an ACK/NACK is expected.)
- Bit 5 shall indicate whether the receiving device shall include its EUI-64 in the ACK/NACK. This setting shall be used whenever an acknowledgement is requested (bit 7 value of 1), and no EUI-64 for the neighbor exists in dlmo.Neighbor. See 9.1.10.1.
- Bit 4 shall indicate the presence or absence of a DAUX sub-header in the DPDU.
- Bit 3 shall indicate whether to include the slow hopping offset in the DMXHR. This value shall be included in unicast DPDUs where slow hopping is used. See 9.1.9.2.4.
- Bit 2 shall indicate whether the transmitting device is a DL clock recipient. This is a signal to the receiver to include a clock correction in the acknowledgement.
- Bits 0-1 shall indicate the DL version number. A value of 00 shall be used, with 1-2 being reserved for future use. A value of 3 ("11") is used in the same location in an ACK/NACK frame and helps to distinguish DPDU from ACK/NACK frames (see 9.3.4).

### 9.3.3.4 Data link layer media access control extension sub header

A DMXHR following the DHDR is summarized in Table 112.

**Table 112 – Data frame DMXHR**

octets	bits								
	7	6	5	4	3	2	1	0	
1 octet	Security control								
1 octet	Crypto key identifier								
0-2 octet	Slow hopping timeslot offset (ExtDlUInt)								

NOTE For future PHYs with more than 16 channels, the channel number may be added as a virtual field.

The length of the DMXHR is 2-4 octets. A DMXHR length of 3 octets, corresponding to a slow hopping rate of about 1.25 s or less, was used to calculate the default dlmo.MaxDsduSize.

As shown in Table 112, attributes include:

- Security control and crypto key identifier. The security fields are left unspecified by the DL, and are set by the security sub-layer. See 9.1.12 for an overview of the relationship between the DL and security sub-layers. While IEEE allows a crypto key identifier as large as 9 octets, its length is always 1 octet in this standard.

- The slow hopping timeslot offset specifies the timeslot offset into the slow hopping period, if necessary to unambiguously identify a timeslot. The presence or absence of this field is indicated in DHDR. The slow hopping timeslot offset is described in 9.1.9.4.9.

### 9.3.3.5 Data link layer auxiliary sub-header

The DAUX sub-header is used for:

- DL neighbor discovery
- Temporarily activating links
- Reporting received signal quality in acknowledgements.

The DAUX sub-header, present only when Bit 4 of the DHDR frame control octet is set to 1, is described in 9.3.5.

A DAUX length of 0 octets was used to calculate the default dlmo.MaxDsduSize.

NOTE dlmo.MaxDsduSize is used to make fragmentation decisions, and the DAUX subheader can be used to activate links in a fragmentation scenario. However, link activation cannot be done during the join process, and as such link activation is never combined with 64-bit addressing. Since the calculation of dlmo.MaxDsduSize includes one 64-bit address, it allows for a DAUX link activation sub-header when a 64-bit address is not used.

### 9.3.3.6 Routing sub-header

There are two variants of the DROUT sub-header. A compressed variant, two octets in length, is used when a single graph is used for addressing. The compressed variant is also used when single-hop routing is used, with the route being implicit in the MAC-level addressing found in the IEEE Std 802.15.4 MHR. When a series of addresses is needed, an uncompressed variant of the DROUT sub-header shall be used.

The DROUT sub-header shall be elided in a DPDU that has no higher order payload, as indicated by a DSDU of zero length.

The compressed variant of the DROUT sub-header shall be used in the common case where a single graph, with an index of 255 or less, is used for routing. It is shown in Table 113.

**Table 113 – DROUT structure, compressed variant**

octets	bits													
	7	6	5	4	3	2	1	0						
1 octet	Compress=1	Priority (Unsigned4)						DIFForwardLimit (Unsigned3)						
0-1 octet	DIFForwardLimitExt (Unsigned8)													
1 octet	GraphID (Unsigned8)													

A DROUT length of 2 octets was used to calculate the default dlmo.MaxDsduSize. MaxDsduSize normally needs to be reduced when source routing is used.

As shown in Table 113, the compressed variant of the DROUT comprises:

- Compress. If this value is set to 1, the compressed variant of the DROUT format shall be used.
- Priority. This shall be set to the DPDU's 4-bit priority.
- DIFForwardLimit and DIFForwardLimitExt (forwarding limit) limit the number of times that a DPDU may be forwarded within a DL subnet. If the forwarding limit is less than 7, the value shall be transmitted in DIFForwardLimit and DIFForwardLimitExt shall be elided. If the forwarding limit is greater than or equal to 7, DIFForwardLimit shall be transmitted as 7, and the forwarding limit shall be transmitted in DIFForwardLimitExt.

The forwarding limit is initialized by the DL when the route is selected, based on the value of dlmo.Route[].ForwardLimit. When a unicast DPDU is successfully received by the DL and needs to be forwarded, the DPDU shall be discarded if its forwarding limit is zero. If its forwarding limit is positive, the forwarding limit shall be decremented (possibly to zero) and the DPDU shall be placed on the message queue.

- GraphID (8 bits). GraphIDs compliant with this standard are 12-bit unsigned integers. In the common case where the route is a single graph ID in the range of 1-255, the compressed variant of the DROUT sub-header shall be used. Additionally, the compressed variant is used in single-hop source routing, wherein GraphID=0 shall indicate that the destination is one hop away. Since the single hop destination address can be found in the MHR, it does not need to be repeated in DROUT. GraphID=0 shall be used during the join process for addressing to and from a neighboring proxy, and is the only way in this standard to indicate a 64-bit destination address in DROUT.

NOTE A DL route may be configured as circular by the system manager, with the DPDU being forwarded until the ForwardLimit decrements to zero. The LH field in the DL header, described in 9.3.3.7, is not intended to prevent circular routes within a DL subnet.

The uncompressed variant of the DROUT sub-header is shown in Table 114.

**Table 114 – DROUT structure, uncompressed variant**

octets	bits													
	7	6	5	4	3	2	1	0						
1 octet	Compress=0	Priority (Unsigned4)					DIFForwardLimit (Unsigned3)							
0-1 octet	DIFForwardLimitExt (Unsigned8)													
1 octet	N (number of entries in routing table; Unsigned8)													
2*N octets	Series of N GraphIDs/addresses (Unsigned16, LSB)													

As shown in Table 114, the uncompressed variant of the DROUT sub-header comprises:

- Compress. If this value is set to 0, the uncompressed variant of the DROUT format shall be used.
- Priority, DIFForwardLimit, and DIFForwardLimitExt are as described above with Table 113.
- N. This field shall be set to the number of entries in Route. The entries may be a combination of GraphIDs and 16-bit DL addresses. The value of N shall not exceed 15.
- Route. This field shall be set to a series of GraphIDs and/or 16-bit DL addresses, specifying the route, in order, along which the DPDU will travel. IETF RFC 4944 limits unicast address ranges to Range 1, 0x x xxxx xxxx xxxx. 12-bit GraphIDs in this field shall be represented as 1010gggggggggg.

When source routing is used, the DROUT sub-header shall be shortened by the DL of intermediate routers as the DPDU proceeds along the route, as described in 9.1.6.

The first entry in the DROUT sub-header is used to determine the next hop. For example, the route may be specified at the source as <000123, 000456, 000789>. The first hop address, <000123>, is used to send the DPDU to an immediate neighbor. The DROUT sub-header, as received by device <000123>, contains the source route <000123, 000456, 000789>. When received, this route is shortened to <000456, 000789> (see 9.1.6.3), indicating that address <000456> is the next hop.

When a graph is specified as the first entry in a source route, the DPDU shall follow that graph until it is terminated, as described in 9.1.6.

### 9.3.3.7 Addressing sub-header

The addressing sub-header (DADDR) includes NL source and destination addresses, along with three NL fields that are visible to the DL.

The DADDR sub-header shall be elided in a DPDU that has no higher order payload, i.e., a DSU of zero length.

The structure of the DADDR sub-header is shown in Table 115.

**Table 115 – DADDR structure**

octets	bits							
	7	6	5	4	3	2	1	0
1 octet	DE	LH	ECN (Unsigned2)		Reserved=0			
1-2 octets	SrcAddr (ExtDUIint)							
1-2 octets	DestAddr (ExtDUIint)							

A DADDR length of 4 octets was used to calculate the default dlmo.MaxDsduSize, reflecting an assumption that one or the other addresses refers to an infrastructure device with an address in the range of 1-127, or a 64-bit with the address encoded as zero.

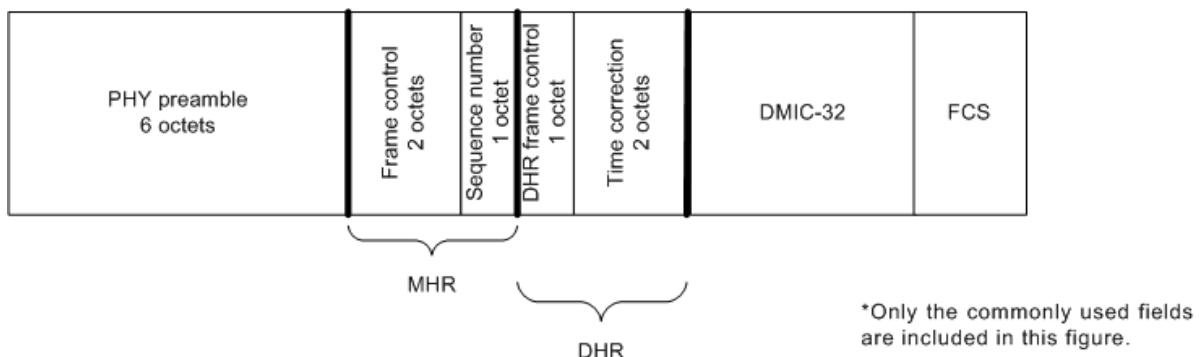
Fields include:

- Discard eligible (DE) shall be set based on the value provided by the NL through DD-DATA.request. DE=1 indicates that the DSDU is eligible to be discarded from the message queue in favor of an incoming DPDU with DE=0, and of equal or higher priority.
- Last hop (LH) shall be set based on the value provided by the network layer through DD-DATA.request. This bit is carried by the DL to avoid circular routes at the network layer, and does not affect DL behavior.
- Explicit congestion notification (ECN) shall be set based on the value provided by the network layer through DD-DATA.request. A router experiencing congestion may set ECN as described in IETF RFC 3168. See 9.1.9.4.5 for a discussion of ECN.
- SrcAddr is set based on the value provided from the network layer through DD-DATA.request. If the 16-bit or 64-bit source address is duplicated in the MHR source address field, SrcAddr shall be set to zero. This covers the case, during the join process, where the source address is a 64-bit address.
- DestAddr is set based on the value provided from the network layer through DD-DATA.request. If the 16-bit or 64-bit destination address is duplicated in the MHR destination address field, DestAddr shall be set to zero. This covers the case, during the join process, where the destination address is a 64-bit address.

By encoding a duplicated 16-bit address as zero, an octet is compressed on the first and last hop when the address is less than 128, providing an energy saving in that case. Processing of a DADDR address encoded as zero is based on the corresponding address in the MHR.

### 9.3.4 Media access control acknowledgement frames

Figure 89 illustrates the structure of ACK/NACK frames.

**Figure 89 – Typical acknowledgement frame layout**

The source address of an ACK/NACK is the address of the device that transmits the ACK/NACK. The destination address is the address of the intended recipient of the ACK/NACK.

Every ACK/NACK frame shall be authenticated with a DMIC, but not encrypted. Some fields are virtual, used in creating the DMIC but not actually transmitted.

ACK/NACK frame uses IEEE Std 802.15.4 data frame type, and can be distinguished from other data frames based on:

- The ACK/NACK frame's timing, following DPDU transmission, as specified in 9.4.3.3; and
- Bits 1-0 of DHR frame control field, as specified with Table 118; and
- A virtual field in the ACK/NACK, echoing the DPDU's original MMIC, as specified with Table 117.

As described in 7.3.2.2, the DMIC in a DL acknowledgement uses the same security policy as the original DPDU, with the exception that the DMIC length shall always be 32-bits regardless of the security policy.

The format of an IEEE Std 802.15.4 MHR is summarized in Table 116.

**Table 116 – Acknowledgement frame MHR**

octets	bits							
	7	6	5	6	3	2	1	0
2 octets	Frame control (octet 0)							
	Frame control (octet 1)							
1 octet	Sequence number							

As shown in Table 116, attributes include:

- Frame control attributes for ACK/NACKs, as follows:
  - Frame type shall be data.
  - Security shall be disabled, as it is handled in the DHR.
  - Frame pending shall be false.
  - Ack.Request shall be false (IEEE Std 802.15.4 does not recognize this as an ACK).
  - Source addressing mode shall be 0x00 (i.e., implicit), except for cases described below where the source address and PAN ID are included in the MHR.
  - Destination addressing mode shall be 0x00 (i.e., implicit).
  - Frame version shall be 0x01.
  - Sequence number, used by the security sub-layer, as described in 7.3.2.4.10. As this standard does not utilize ack frame type specified by IEEE Std 802.15.4, ACK/NACK frame does not carry the sequence number from the DPDU, rather the sequence number shall be in accordance with that of a data frame (see 9.3.3.2).

The acknowledger's EUI-64 shall be included in the source address field of the ACK/NACKs MHR if so requested in the received DPDU's DHDR. Normally, the 16-bit DL source address of the ACK/NACK is not transmitted, because it matches the destination address of the received DPDU. However, in duocast or n-cast acknowledgements, one or more acknowledgements may be sent by different devices. Therefore, in cases where the acknowledger's address is different from the destination address of the received DPDU, the acknowledgement shall also include the acknowledger's source PAN ID and 16-bit DL source address in the MHR, with the assumption that the acknowledger's EUI-64 is already known by the recipient of the acknowledgement.

A prototype DHR following a MHR is summarized in Table 117.

**Table 117 – Acknowledgement frame DHR**

octets	bits							
	7	6	5	4	3	2	1	0
1 octet	DHR ACK/NACK Frame control							
4 octets (virtual)	Echoed DMIC of received DPDU							
0-2 octets	Time correction (Unsigned16, LSB) When requested							
0-2 octets	Timeslot offset (ExtDIUInt) When needed							
0-3 octets	DAUX sub-header Usually absent							

As shown in Table 117, attributes include:

- The DHR ACK/NACK frame control octet is described in Table 118.
- Echoed DMIC of received DPDU. For a discussion of handling of this virtual field, see 7.3.2. To unambiguously connect the ACK/NACK with the DPDU, the DMIC of the DPDU is included in the ACK/NACK's DHR as a virtual field, with octet ordering matching the DPDU's DMIC. This virtual field is used to calculate the ACK/NACK's DMIC, but not transmitted. If the received DMIC is longer than 4 octets, only the initial (leftmost) 4 octets of the DMIC are echoed as a virtual field.
- Time correction (LSB). Used by DL clock sources to correct the time of the DL clock recipient, if it is requested in the received DPDU's DHDR. This 2-octet unsigned value, when included in the ACK/NACK, echoes the time that the DPDU was received. The value, in  $2^{-20}$  s (approximately 0.954  $\mu$ s), reports an offset from the scheduled start time of the current timeslot in the acknowledger's time base. The reported value is based on DPDU's start time. See 9.1.9.3.2.
  - Acknowledger's timeslot offset is provided, when needed, within a slow hopping period. This one-octet value, when included in the NACK/ACK, indicates the current timeslot in the acknowledger's time base. It shall be included only when the received DPDU is received in a different slow hopping timeslot than is used for the acknowledgement. The first timeslot in a slow hopping period has an offset of zero. When the corrected timeslot offset is nonzero, the time correction (previous field), when included, shall be an offset of the corrected scheduled timeslot time. Security requires that a device's time increases from timeslot to timeslot. Therefore, if the timeslot is corrected to an earlier timeslot by a clock recipient, there shall be an interruption in service, equal to the magnitude of the timeslot correction plus at least one timeslot. See 9.1.9.4.9.
  - Auxiliary sub-header (DAUX). DAUX may be included in an ACK/NACK, for the limited purpose of echoing received signal quality (see 9.3.5.5).

In an ACK/NACK DPDU, the DHR frame control octet communicates the ACK/NACK selections, as shown in Table 118.

**Table 118 – DHR ACK/NACK frame control**

octets	bits							
	7	6	5	4	3	2	1	0
1 octet	Include clock correction 0= no 1= yes	Include slow-hopping timeslot offset 0= no 1= yes	ACK/NACK type 0= ACK 1=ACK/ECN 2=NACK0 3=NACK1	Auxiliary sub-header 0= no DAUX 1= DAUX included	Reserved=0	Reserved=11		

The DL protocol version number and MAC security key always match the received DPDU, and therefore are not indicated in the ACK/NACK.

Bit content is as follows:

- Bit 7 shall indicate whether the ACK/NACK includes clock correction information.
- Bit 6 shall indicate whether the ACK/NACK includes a slow hopping offset.
- Bits 5 and 4 shall indicate whether the PDU is an ACK or a NACK, as follows:
  - 00 is an acknowledgement.
  - 01 is an acknowledgement with an explicit congestion notification (ECN). See 9.1.9.4.5. A router that is signaling ECN bits in the forward direction should also signal the ECN through DL acknowledgements, if the priority of the DPDU is 7 or less. A device receiving an ECN through a DL acknowledgement may treat this signal as early notification that it is likely to receive an ECN at upper layers.
  - 02 is a NACK0, signaling that the DPDU was received but could not be acknowledged due to message queue congestion. See 9.1.9.4.4.
  - 03 is a NACK1, signaling that the DPDU was received but was not accepted due to recent history of forwarding problems along the route. See 9.1.9.4.4.
- Bit 3 shall indicate whether the ACK/NACK includes a DAUX sub-header. DAUX may be included in an ACK/NACK for the limited purpose of reporting received signal quality.
- Bit 2 is reserved and shall be set to zero.
- Bits 1-0 are reserved and shall be set to ones (11) to distinguish ACK/NACK frames from other DPDU frames.

### **9.3.5 Data link layer auxiliary sub-header**

#### **9.3.5.1 General**

An auxiliary sub-header (DAUX) may be included in any DPDU or ACK/NACK. The first 3 bits of the DAUX determine its type, and the subsequent sub-header format is different for each type. Types include:

- Advertisement (type 0): Provides information needed by new devices to synchronize with the DL subnet, and to join the network. DPDU only.
- Solicitation (type 1): Solicits an advertisement from a neighboring device. DPDU only.
- Activate link (type 2): Activates an idle link for a period of time. DPDU only.
- Signal quality (type 3): Echo received signal quality, in ACK/NACK only.

Types 4-7 are reserved for future use.

NOTE Following DL header conventions, DAUX fields use LSB (little-endian) order for transmission. There are some similar structures in the DLMO that use MSB (big-endian) order. For example, superframe structures are specified in both places, using LSB in the DL header and MSB in DLMO attributes.

#### **9.3.5.2 Advertisement auxiliary sub-header**

##### **9.3.5.2.1 General**

Fields within an advertisement DAUX can be grouped logically as:

- Advertisement selections;
- Time synchronization;
- Superframe information;
- Join information; and
- Integrity check.

Table 119 summarizes the structure of the advertisement DAUX.

**Table 119 – Advertisement DAUX structure**

octets	bits							
	7	6	5	4	3	2	1	0
1 octet	Advertisement selections; see Table 120							
6 octets	Time synchronization; see Table 122							
6-10 octets	Superframe information; see Table 124							
4-10 octets	Join information; see Table 127							
2 octets	Integrity check (MSB); see 9.3.5.2.4.3							

NOTE As described in 9.3.5.2.4.2, join information field length is limited to 10 octets.

The advertising routers subnet ID and 16-bit DL address are conveyed through the MAC sub-layer, and do not need to be transmitted redundantly within the DAUX.

An advertisement DAUX may be included within a DPDU, but shall not be included within an ACK/NACK.

An advertisement includes information that enables the DL to create superframes and links to be used during the join process. This information shall be retained by the DL at the end of the join process and, along with DL defaults, constitute a starting position for the DL. The same links used for joining are temporarily used for general communications until the system manager provides an alternative configuration.

Attributes set by the DL based on information received in the advertisement include:

- dlmo.SubnetID is set based on the SubnetID in the advertisement.
- TAI time is synchronized by the advertisement.
- dlmo.Superframe #1 is created by the DL with fields copied from the advertisement.
- dlmo.Link #1 is created as a transmit link by the DL with fields copied from the advertisement.
- dlmo.Link #2 is created as a receive link by the DL with fields copied from the advertisement.
- dlmo.Link #3 may be created as passive scanning receive links by the DL with fields copied from the advertisement if provided.
- dlmo.Neighbor is initialized by the DL with an entry corresponding to the advertising router.
- dlmo.Graph #1 is automatically created by the DL, to referring to the advertising router.
- dlmo.Route #1 is automatically created by the DL as the default route using Graph #1.

### 9.3.5.2.2 Advertisement selections

Table 120 specifies the advertisement selections field in the advertisement DAUX.

**Table 120 – Advertisement selections elements**

Element name	Element encoding
DauxType	Type: Unsigned3 0=advertisement DAUX
ChMapOv	Type: Unsigned1 0=default
DauxOptSlowHop	Type: Unsigned1 0=default
Reserved (octet alignment)	Type: Unsigned3=0

Table 121 illustrates the structure of the advertisement selections field.

**Table 121 – Advertisement selections**

octets	bits							
	7	6	5	4	3	2	1	0
1 octet	DauxType=0			DauxChMapOv	DauxOptSlowHop		Reserved=0	

The length of the advertisement selections field is 1 octet. As shown in Table 120, attributes include:

- DauxType. Always set to 0 for an advertisement DAUX. Indicates the DAUX structure in Table 119.
- DauxChMapOv. 1 indicates that the DauxChMap field is transmitted. 0 indicates the default of 0xFFFF. This field corresponds to dlmo.Superframe[].ChMapOv.
- DauxOptSlowHop. 1 indicates that slow hopping is being used, and DauxChRate is transmitted. 0 indicates the default of slotted hopping.
- Bits 2-0 are reserved and shall be set to 0.

### 9.3.5.2.3 Advertisement time synchronization

Table 122 specifies the time synchronization field in the advertisement DAUX.

**Table 122 – Advertisement time synchronization elements**

Element name	Element encoding
DauxTAIsecond (current TAI time)	Type: Unsigned32 (LSB) Units: 1 s
DauxTAIfraction (fractional TAI second)	Type: Unsigned16 (LSB) Units: $2^{-15}$ s

Table 123 illustrates the structure of the advertisement time synchronization field.

**Table 123 – Advertisement time synchronization structure**

octets	bits							
	7	6	5	4	3	2	1	0
4 octets	DauxTAIsecond (LSB)							
2 octets	DauxTAIfraction (LSB)							

The length of the time synchronization field is 6 octets. As shown in Table 122, subfields include:

- DauxTAIsecond. Current TAI time in units of 1 s.
- DauxTAIfraction. Fractional TAI second in units of  $2^{-15}$  s, with a range of 0 to 32667. Within the TAI second, this indicates the advertisement DPDU's actual start time. (An implementation that actually clocks based on SFD timing should account for a DPDU start time that is nominally 1 octet, or 32  $\mu$ s, later than the time that the SFD is completely transmitted/received.)

See 9.1.9 for more information on TAI time and timeslot alignment.

The identity and scheduled timing of the current timeslot can be derived from the DPDU's actual start time combined with the join superframe description. See 9.1.9.1.5.

The time in an advertisement shall be an accurate reflection of the advertiser's internal TAI clock to within  $\pm 96$   $\mu$ s (3 octets), which is the general transmission window for the standard.

DL acknowledgement to a join request includes a clock correction that may be more precise than the original advertisement, and also more current.

### 9.3.5.2.4 Advertisement join superframe and links

#### 9.3.5.2.4.1 Advertisement join superframe

There are three links specified by the advertisement related to neighbor discovery:

- Link #1 for sending join requests, addressed to the neighboring advertising router;
- Link #2 for receiving subsequent join responses from the advertising router; and
- Link #3 for scanning for additional neighbors after the device successfully joins the network.

All of these links refer to superframe #1, which is also specified by the advertisement.

Field names in the advertisement correspond to equivalent fields in dlmo.Superframe and dlmo.Link. Following DL header conventions, LSB octet ordering is used on certain fields that are transmitted using MSB ordering in the superframe itself. To minimize processing requirements and to compress the DAUX sub-header, a subset of superframe and link features is supported through the advertisement.

Table 124 specifies the join superframe information field.

**Table 124 – Join superframe information subfields**

Subfield name	Subfield encoding
DauxTsDur (timeslot duration)	Type: Unsigned16 Units: $2^{-20}$ s
DauxChIndex (channel hopping pattern ID)	Type: ExtDLUInt Valid value set: 1-5
DauxChBirth (Channel hopping reference starting point)	Type: Unsigned8 Valid value set: 0-255
DauxSfPeriod (number of timeslots in each superframe cycle)	Type: ExtDLUInt
DauxSfBirth (Superframe cycle starting point)	Type: ExtDLUInt Valid value set: 0-127
DauxChRate (length of each slow hopping period, in number of timeslots)	Type: Unsigned8 Not transmitted and defaults to 1 if DauxOptSlowHop=0
DauxChMap (channel hopping channel map for spectrum management)	Type: Unsigned16 (LSB) Not transmitted and defaults to 0xFFFF if advChMapOv=0

Table 125 summarizes the structure of the join superframe information field in the advertisement DAUX. ExtDLUInt fields are shown as one octet.

**Table 125 – Join superframe information structure**

octets	bits								
	7	6	5	4	3	2	1	0	
2 octets	DauxTsDur (LSB)								
1 octet	DauxChIndex								
1 octet	DauxChBirth								
1-2 octets	DauxSfPeriod								
1 octet	DauxSfBirth								
0-1 octet	DauxChRate								
0-2 octets	DauxChMap (LSB)								

The length of the join superframe information field is 6-10 octets.

Each subfield of advertisement join superframe information corresponds to a field in dlmo.Superframe. See 9.4.3.5.

When creating superframe #1 from the advertisement, the DL uses values from the advertisement to initialize corresponding superframe fields. Fields in the superframe #1 that

do not have equivalently named fields in the advertisement default to fixed values. Table 126 shows the mapping from the advertisement's subfields to superframe #1.

**Table 126 – Superframe derived from advertisement**

Superframe field name	Value	Notes
* Index	1	
TsDur	DauxTsDur	
ChIndex	DauxChIndex	
ChBirth	DauxChBirth	
SFType	0	
Priority	0	
ChMapOv	DauxChMapOv	
IdleUsed	0	
SfPeriod	DauxSfPeriod	Compressed data type used in advertisement. Limits superframe used for joining to approximately 5 minutes.
SfBirth	DauxSfBirth	Compressed data type used in advertisement, consistent with SfPeriod.
ChRate	DauxChRate	Compressed data type used in advertisement. Limits superframes used for joining to a slow hopping rate of approximately 2.5 seconds.
ChMap	DauxChMap	Convert LSB to MSB.
IdleTimer	null	
rndSlots	null	

#### 9.3.5.2.4.2 Advertisement join links

There are two sets of links related to joining provided in every advertisement:

- An outbound set of links for transmitting join requests, used to initialize dlmo.Link #1; and
- An inbound set of links for receiving join responses, used to initialize dlmo.Link #2.
- In addition, the advertising router may provide a set of links that a device can use to scan for advertisements when joining is complete, used to initialize dlmo.Link #3 when provided.
- Links used for joining are constrained to a basic set of features. Default timeslot templates are used; see Table 165, Table 166, and Table 167.

Three types of links are identified:

- JoinTx links are used for transmitting join requests to the advertising router.
- JoinRx links are monitored while waiting for a join response.
- AdvRx links, when provided, are activated when joining is complete to passively scan for advertisements from alternative routers.

Table 127 specifies the join information field in the advertisement DAUX.

**Table 127 – Join information elements**

Element name	Element encoding
DauxJoinBackoff (maximum extent of backoff and retry while joining)	Type: Unsigned4
DauxJoinTimeout (join timeout)	Type: Unsigned4
DauxJoinFIxmit (indicates fields that are transmitted)	Type: Unsigned8
DauxJoinTx (JoinTx link(s))	Type: See Table 184
DauxJoinRx (JoinRx link(s))	Type: See Table 184
DauxAdvRx (Advertisement link(s))	Type: See Table 184 (or null)

The element DauxJoinFldXmit selects the link scheduling parameters used for elements DauxJoinTx, DauxJoinRx, and DauxAdvRx. DauxJoinFldXmit values 0 to 3 correspond to the semantics described in Table 184.

Table 128 illustrates the structure of the join information field. Fields DauxJoinTx, DauxJoinRx, and DauxAdvRx may be 1 to 4 octets in length (or null only for DauxAdvRx), depending on the configuration and selection as described in Table 184.

**Table 128 – Join information structure**

octets	bits									
	7	6	5	4	3	2	1	0		
1 octet	DauxJoinBackoff	DauxJoinTimeout								
1 octet	DauxJoinFldXmit									
1-4 octets	DauxJoinTx									
1-4 octets	DauxJoinRx									
0-4 octets	DauxAdvRx (may be absent)									

Depending on the options selected, it would appear from Table 128 that the length would be in the range of 4-14 octets. However, selections shall be configured such that the total length of fields shown in Table 128 does not exceed 10 octets.

As shown in Table 128, attributes include:

- DauxJoinBackoff. Maximum extent of exponential backoff on joining. If a join request does not receive a DL ACK due to CCA or a failed transmission, the device shall back off by selecting a random time in the range of 0 to 1 s for the first retry, and use the first available JoinTx timeslot after that time. Then double the time range with each retry. The device may retry up to DauxJoinBackoff times and shall not retry more than DauxJoinBackoff times. At that point, the device should abort the attempt to send a message to the advertising router, revert to the provisioned state, and search for another advertisement.
- DauxJoinTimeout. Guaranteed time, in s, to receive a system manager response to a join request. Expressed as an exponent, to the power of 2. For example, if DauxJoinTimeout=5, then the device can expect completion within  $2^5=32$  s. Following a timeout, the device should abort the attempt to join through the advertising router, revert to the provisioned state, and search for another advertisement.
- DauxJoinFldXmit:
- Bits 7, 6: Unsigned2, describing contents of DauxJoinTx. See Table 184. Corresponds to dlmo.Link[].SchedType for link #1. Supported range is 0-2.
- Bits 5, 4: Unsigned2, describing contents of DauxJoinRx. See Table 184. Corresponds to dlmo.Link[].SchedType for link #2. Supported range is 0-2.
- Bit 3: If Bit3=1, transmit DauxAdvRx. If Bit3=0, DauxAdvRx is null and not transmitted.
- Bits 2, 1: Unsigned2, describing contents of DauxAdvRx. See Table 184. Supported range is 0-2. Corresponds to dlmo.Link[].SchedType for link #3. If Bit3=0, Bits 2 and 1 are meaningless and shall also be 0.
- Bit 0 is reserved and shall be set to zero.
- DauxJoinTx. The join transmission timeslot(s) in each superframe cycle, corresponding to dlmo.Link[].Schedule for link #1. These are the transmission opportunities in which to send join requests. Bits 7, 6 of DauxJoinFldXmit specify the format of DauxJoinTx, as defined in Table 184.
- DauxJoinRx. The join receive timeslot(s) in each superframe cycle, corresponding to dlmo.Link[].Schedule for link #2. Bits 5, 4 of DauxJoinFldXmit specify the format of DauxJoinRx as defined in Table 184.
- DauxAdvRx. Receive links, for scanning for additional neighbors after joining, corresponding to dlmo.Link[].Schedule for link #3 when provided. Bits 2, 1 of DauxJoinFldXmit specify the format of DauxAdvRx as defined in Table 184. It is

transmitted and meaningful only when DauxJoinFldXmit.Bit 3=1; otherwise its value is null and no corresponding links are created in Table 129.

Links are added to dlmo.Link[] based on parameters in the advertisement. Fields are set as shown in Table 129.

**Table 129 – Defaults for links created from advertisements**

Field name	DauxJoinTx	DauxJoinRx	DauxAdvRx (if DauxJoinFldXmit.Bit3=1)
* Index	1	2	3
SuperframeIndex	1	1	1
Type-Transmit	1	0	0
Type-Receive	0	1	1
Type-Exponential Backoff	1	0	0
Type –Idle	0	0	1
Type-Discovery	0	0	0
Type-JoinResponse	0	0	0
Type-SelectiveAllowed	1	1	1
Template1	2	1	3
Template2	Null	Null	Null
NeighborType	1	0	0
Graph Type	0	0	0
SchedType	From advertisement DauxJoinFldXmit Bits 7-6	From advertisement DauxJoinFldXmit Bits 5-4	From advertisement DauxJoinFldXmit Bits 2-1
ChType	0	0	0
PriorityType	0	0	0
Neighbor	Address of advertising neighbor	Null	Null
GraphID	Null	Null	Null
Schedule	From advertisement DauxJoinTx	From advertisement DauxJoinRx	From advertisement DauxAdvRx
ChOffset	Null	Null	Null
Priority	Null	Null	Null

Link #3, intended to be used to scan for neighbors after joining, is configured as an idle link. Normally, idle links are enabled through a DAUX subheader as described in 9.3.5.4. In addition, when the DL changes to the join state, it shall activate Link #3 for a period of time equal to the initial value of dlmo.DiscoveryAlert.Duration (default 60 s). This causes the DL to collect information into the dlmo.Candidates table and then report it to system manager through the NeighborDiscovery alert, unless the DL is reconfigured by the system manager during the interval for a different result.

NOTE 1 GraphType in Table 129 is set to zero, indicating that the feature is not applicable in this context. See 9.4.3.7.2.

The links created from the advertisement also need entries in dlmo.Neighbor, dlmo.Graph, and dlmo.Route. These entries are automatically added by the DL at the same time as the links, with values as shown in Table 130, Table 131, and Table 132 below.

**Table 130 – dlmo.Neighbor entry created from advertisements**

Field name	Value
* Index	Address of advertising router
EUI-64	Acquired from the advertising router as described in 9.1.10.1
GroupCode	0
ClockSource	2
ExtGrCnt	0
DiagLevel	0b01
LinkBacklog	0
ExtendGraph	Null
LinkBacklogIndex	Null
LinkBacklogDur	Null

NOTE 2 ExtendGraph in Table 130 is set to null, indicating that the feature is not applicable in this context. See 9.4.3.4.2.

**Table 131 – dlmo.Graph entry created from advertisements**

Field name	Value
* Index	1
PreferredBranch	0
NeighborCount	1
Queue	0
MaxLifetime	0
Neighbors	Address of advertising router

**Table 132 – dlmo.Route entry created from advertisements**

Field name	Value
* Index	1
Size	1
Alternative	3
ForwardLimit	16
Route	One entry: 0xA001 (Graph #1)
Selector	Null

NOTE 3 Route information in Table 132 is intended to be used as the default route after the device joins the network. Join messages to the neighboring proxy use source routing as described in 9.3.3.6. Sample DL headers for join messages are provided in Annex T.

DLMO updates from DAUX join information are made at two points in the DL life cycle. First, when a device in the default state receives an advertisement from a provisioning mini-network (SubnetID=1), it needs to update DLMO attributes with DAUX join information in order to join the mini-network. Second, when a device in the provisioned state joins a target network via an advertising router, it needs to update the DLMO with DAUX join information in order to join the target network.

There are various other times when a DL might receive and process advertisements, such as when searching for multiple candidate neighbors in the provisioned state, searching for candidate neighbors in the joined state, or receiving DL subnet time updates in any state. The receipt and processing of such advertisements may trigger a join request only in the default or provisioned state, and DAUX join information in the advertisement is posted to the recipient's DLMO only when the device attempts to join a mini-network from the default state or attempts to join a target network from the provisioned state.

### 9.3.5.2.4.3 Integrity check

DPDUs that embed a DAUX include the IEEE 16-bit ITU-T CRC (FCS) as an integrity check on the overall MPDU, plus a DMIC for authentication as a further integrity check. However,

the DMIC cannot be authenticated without a shared sense of time, a shared security key, and knowledge of the senders EUI-64.

NOTE 1 Time within the DAUX may be used as a shared sense of time for authentication of the DPDUs that contains the DAUX.

This standard permits a device to view and use the DAUX even if it cannot authenticate the overall DPDUs. It was deemed insufficient to rely on the IEEE Std 802.15.4 FCS as the only integrity check for advertisements. For this reason, an additional 16-bit integrity check is included within the DAUX, covering only the contents of the DAUX itself.

The DAUX integrity check is similar to the UDP checksum described in IETF RFC 768. The checksum is the 16-bit ones complement of the ones complement sum of the octets that comprise the DAUX sub-header, excluding the integrity check itself, padded with zero octets at the end (if necessary) to make a multiple of two octets. Octet ordering is as transmitted. If the computed checksum is zero, it is transmitted as all ones. An all-zero transmitted checksum value means that the transmitter generated no checksum.

Transmission order of the integrity check is MSB, i.e., with the first octet reflecting bit operations on odd-numbered octets, with the octet count starting at 1, in the DAUX sub-header (octets 1, 3, 5, etc.) and the second octet from even-numbered octets in the DAUX sub-header (octets 2, 4, 6, etc.).

NOTE 2 The use of a secret subnet security key for advertisements enables those advertisements to be trusted after the device has joined the DL subnet. Advertisements may be used for periodic surveys of neighboring routers, or for periodic time updates by devices with low duty cycles.

NOTE 3 The advertisement provides only the 16-bit DL address of the advertising router. An EUI-64 is needed for subsequent exchange of DPDUs with that router. As described in 9.1.14.2, the DL shall acquire the EUI-64 from the advertising device.

### **9.3.5.2.5 Configuring advertisements**

The timing of advertisements is determined by the structure of the advertising device's superframes and links. Any link may include an advertisement flag, which indicates that the advertisement is included in the DAUX.

An index value in the attribute dlmo.AdvSuperframe (see Table 141) selects a superframe in dlmo.Superframe that shall be used as a reference to build the advertisement. The reference superframe is configured by the system manager by establishing a superframe, which may be idle, and referring to its index in the dlmo.AdvSuperframe attribute. The reference superframe shall not use features that cannot be represented in the join superframe information field in Table 124.

A zero value in dlmo.AdvSuperframe is the default, and indicates that the advertisement has not been configured.

Link information is placed in the dlmo.AdvJoinInfo attribute, in exactly the format that it is transmitted in the advertisement in the position corresponding to Table 128. In this way, the new devices JoinTx, Join Rx, and AdvRx links are specified.

The system manager configures superframes in the advertising router that match the advertisement. At the time of JoinTx links, the advertising router shall be configured with links to receive DPDUs. At the time of JoinRx links, the advertising router shall be configured with links where JoinResponse=1 (see Table 182).

### **9.3.5.3 Solicitation auxiliary sub-header**

#### **9.3.5.3.1 General**

A solicitation is a request for an advertisement to be transmitted by an active scanning host in range, on the same channel as the solicitation itself. (See 9.1.13.4.)

Attributes within a solicitation DAUX can be grouped logically as:

- Solicitation header; and

- Subnet ID.

The solicitation does not have a reliable sense of time, nor does it necessarily have a secret security key. Therefore, to allow the receiver of the solicitation to decode its DMIC, a solicitation's DMIC shall be built using a security key of K\_global and a nominal TAI time of zero. This allows for consistent processing, and provides a strong integrity check for the DPDU. No additional integrity check is included in the solicitation's DAUX.

A solicitation's MHR (IEEE MAC header) shall not provide a source or destination address, and it shall not specify a subnet. See 9.1.5.

### **9.3.5.3.2 Solicitation fields**

Table 133 specifies the solicitation header in the solicitation DAUX.

**Table 133 – Solicitation header subfields**

Subfield name	Subfield encoding
DauxType	Type: Unsigned3 1= solicitation DAUX
DauxSubnetInclude (indicates whether to transmit DauxSubnetID in solicitation)	Type: Unsigned1
Reserved	Type: Unsigned4=0

The length of the solicitation header is 1 octet. As shown in Table 133, elements include:

- DauxType. Set to 1 to indicate a solicitation DAUX.
  - DauxSubnetInclude. Indicates whether to transmit the DauxSubnetID field in the solicitation. If DauxSubnetInclude=0, the DauxSubnetID field is not transmitted, and the receiver (active scanning host) shall use the default value of DauxSubnetID=0 in filtering.

Table 134 illustrates the structure of the solicitation header.

**Table 134 – Solicitation header structure**

octets	bits									
	7	6	5	4	3	2	1	0		
1 octet	DauxType=1			DauxSubnetInclude	Reserved=0					

Table 135 specifies the other fields in the solicitation DAUX.

**Table 135 – Solicitation DAUX fields**

<b>Field name</b>	<b>Field encoding</b>
DauxSubnetID (specifies the SubnetID)	Type: Unsigned16 (LSB)

DauxSubnetID transmits a subnet ID that can be used as a filter by the receiver, based on the receiver's dlmo.SolicFilter attribute. When DauxSubNetInclude=0, DauxSubnetID defaults to 0x0000 and is not transmitted.

Table 136 summarizes the structure of the solicitation DAUX.

**Table 136 – Solicitation DAUX structure**

### 9.3.5.3.3 Configuring solicitations

Due to regulatory and safety requirements, some applications cannot tolerate devices that transmit DPDUs while they are idle or in transit. Therefore, the default in this standard does not include solicitations in its configuration. The intent is that devices will be configured with solicitations, as appropriate, when they are provisioned or subsequently in the life cycle.

The timing of solicitations is determined by the structure of the device's superframes and links. Transmission of a solicitation is triggered by a `dlmo.Link[].Discovery` field set to a value of 3.

When a solicitation is transmitted, the contents of `dlmo.SolicTemplate` shall be copied verbatim into the DAUX subheader. If the length of `SolicTemplate` is zero, this shall be interpreted as a configuration error and the link shall be ignored.

To support regulatory and safety requirements, solicitations can be enabled and disabled on a timed basis by the system manager through the attributes `dlmo.RadioSilence`, `dlmo.RadioSleep`, and `dlmo.Superframe[].IdleTimer`.

### 9.3.5.4 Activate link auxiliary sub-header

#### 9.3.5.4.1 General

The activate link DAUX provides a mechanism that enables a transmitter to activate idle timeslots for a short period of time in order to efficiently forward a backlog of messages that have accumulated in a transmitter's message queue. These idle links, when so configured by the system manager, are activated by the router in response to a burst of messages flowing through a DL toward a particular neighbor.

The system manager configures:

- An idle transmit link on the transmission side, addressed to a particular neighbor or group of neighbors, with a particular link index and schedule.
- An idle receiver link on the reception side, with the same link index and schedule.
- A set of parameters in the neighbor table, indicating the link index (`LinkBacklogIndex`), the size of the backlog that should trigger link activation (`LinkBacklog`), and the duration of link activation (`LinkBacklogDur`). See 9.4.3.4.2 for the definition of these parameters.

When the transmitter of a DPDU detects that there are `LinkBacklog` DPDUs on the message queue that can be forwarded to the DPDU's destination address, the transmitter should use the activate link auxiliary sub-header to activate the idle receive link through the activate link auxiliary sub-header. When the transmitter receives an acknowledgement of the DPDU, that implies that the message has been processed and that the receive side of the idle link has been activated. The transmitter should then activate the transmit side of the idle link for the designated number of communication opportunities.

The activated transmit link might be addressed to a group of neighbors, however, the receive side of the activated link occurs on only one neighbor. Therefore, only DPDUs addressed to that neighbor should be considered as candidates for the activated link.

The activate link DAUX provides a link index and a number of communication opportunities that is used to activate an idle link by the receiver of the DPDU. It has the result of immediately activating an idle link for reception, for the number of communication opportunities indicated by `DauxActivateDur`. The transmitter of the activate link message is, in essence, informing the receiver that queued messages will be following that will be sent during communication opportunities (timeslots) associated with a particular receive link.

Activation of idle links is triggered on the transmitter side by multiple queued DPDUs that can be routed to the receiver. See 9.4.3.4.2 for a description of the transmit side of the activate link message (`LinkBacklogIndex`, `LinkBacklogDur`).

### 9.3.5.4.2 Fields

Table 137 summarizes the activate link DAUX.

**Table 137 – Activate link DAUX fields**

Field name	Field encoding
DauxType	Type: Unsigned3 2=Link activation DAUX
Reserved (octet alignment)	Type: Unsigned5=0
DauxLink_ID (identifier for link)	Type: ExtDLUInt
DauxActivateDur (number of communication opportunities (timeslots) to activate the link, link occurrences)	Type: Unsigned8

The link is activated for the number of occurrences of the link, whether the link is used or not. For example, the link occurrence is counted even if it is not used because of a higher priority link in the same timeslot. The link activation period begins with the next full timeslot after the link activation DAUX is received. See 9.4.3.4.2.

Table 138 illustrates the structure of the activate link DAUX field.

**Table 138 – Activate link DAUX structure**

octets	bits							
	7	6	5	4	3	2	1	0
1 octet	DauxType				Reserved=0			
1 or 2 octets	DauxLinkID							
1 octet	DauxActivateDur							

### 9.3.5.5 Signal quality auxiliary sub-header

#### 9.3.5.5.1 General

The signal quality DAUX reports the quality of the received signal in an acknowledgement, to support collection of round-trip signal quality diagnostics. Two octets are reported, one for signal strength (RSSI) and one for signal quality (RSQI). RSSI and RSQI are described in 9.1.15.2.

#### 9.3.5.5.2 Fields

Table 139 summarizes the reporting of received signal quality DAUX.

**Table 139 – Reporting received signal quality DAUX fields**

Field name	Field encoding
DauxType	Type: Unsigned3 3=Signal quality DAUX
Reserved (octet alignment)	Type: Bit5=0
DauxRSSI (RSSI)	Type: Integer8
DauxRSQI (RSQI)	Type: Unsigned8

Table 140 illustrates the structure of the report received signal quality DAUX.

**Table 140 – Report received signal quality DAUX structure**

octets	bits							
	7	6	5	4	3	2	1	0
1 octet	DauxType				Reserved=0			
1 octet	DauxRSSI							
1 octet	DauxRSQI							

## **9.4 Data link layer management information base**

### **9.4.1 General**

For information on the general handling of standard management objects in the DL, see 9.1.11.

### **9.4.2 Data link layer management object attributes**

#### **9.4.2.1 General**

Table 141 summarizes the data link layer management object (DLMO) attributes. OctetStrings with a length of zero are referred to as null.

**Table 141 – DLMO attributes**

<b>Standard object type name: dlmo (DL management object)</b>				
<b>Standard object type identifier: 124</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
ActScanHostFract	1	Device's behavior as an active scanning host	Type: Unsigned8	See 9.4.2.2
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0	
			Valid value set: 0 – 255	
AdvJoinInfo	2	Join information to be placed in advertisement	Type: OctetString	See 9.4.2.3
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Null	
			Valid value set: N/A	
AdvSuperframe	3	Superframe reference for advertisement	Type: Unsigned16 (MSB)	See 9.4.2.3
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0	
			Valid value set: 0 – 32767	
SubnetID	4	Identifier of the subnet that the DL has joined or is attempting to join	Type: Unsigned16 (MSB)	See 9.4.2.4
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set: 0 – 0xFFFF	
SolicTemplate	5	Template of DAUX subheader used for solicitations	Type: OctetString	See 9.4.2.5
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Null	
			Valid value set: N/A	
AdvFilter	6	Filter used on incoming advertisements during neighbor discovery	Type: OctetString See Table 142	See 9.4.2.20
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: See 9.4.2.20	
			Valid value set: N/A	
SolicFilter	7	Filter used on incoming solicitations.	Type: OctetString See Table 142	See 9.4.2.20
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: See 9.4.2.20	
			Valid value set: N/A	
TaiTime	8	TAI time for DL	Type: TAINetworkTimeValue	Units: 2 <sup>-16</sup> s See 9.4.2.6
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
TaiAdjust	9	Adjust TaiTime at an instant that is scheduled by the system manager	Type: OctetString; see Table 143	See 9.4.2.21
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Null	
			Valid value set: N/A	
MaxBackoffExp	10	Maximum backoff exponent for retries	Type: Unsigned8	See 9.4.2.7
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 5	
			Valid value set: 3 – 8	

<b>Standard object type name: dlmo (DL management object)</b>				
<b>Standard object type identifier: 124</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
MaxDsduSize	11	Maximum octets that can be accommodated in a single DSDU	Type: Unsigned8	See 9.4.2.8
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 96	
			Valid value set: 76 – 96	
MaxLifetime	12	Maximum DPDU lifetime	Type: Unsigned16 (MSB)	Units: ¼ s See 9.4.2.9
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 120 (30 s)	
			Valid value set: 8 – 1920 (2 s – 480 s)	
NackBackoffDur	13	Duration of backoff after receiving a NACK	Type: Unsigned16 (MSB)	Units: ¼ s See 9.4.2.10
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 60 (15 s)	
			Valid value set: 8 – 1920 (2 s – 480 s)	
LinkPriorityXmit	14	Default priority for transmit links	Type: Unsigned8	See 9.4.2.11
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 8	
			Valid value set: 0 – 15	
LinkPriorityRcv	15	Default priority for receive links	Type: Unsigned8	See 9.4.2.11
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0	
			Valid value set: 0 – 15	
EnergyDesign	16	DL's energy capacity as designed	Type: OctetString	See 9.4.2.22
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: See 9.4.2.22	
			Valid value set: N/A	
EnergyLeft	17	Remaining energy for DL	Type: Integer16	See 9.1.17
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
DeviceCapability	18	Device capabilities	Type: OctetString See Table 147	See 9.4.2.23
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: See 9.4.2.23	
			Valid value set: N/A	
IdleChannels	19	Radio channels that shall be idle	Type: Unsigned16	See 9.4.2.12
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0	
			Valid value set: 0 – 0xffff	

<b>Standard object type name: dlmo (DL management object)</b>				
<b>Standard object type identifier: 124</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
ClockExpire	20	Clock expiration deadline.	Type: Unsigned16(MSB)	Units 1 s See 9.4.2.13
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: See description	
			Valid value set:	
ClockStale	21	DL clock source timeout	Type: Unsigned16 (MSB)	Units: 1 s See 9.4.2.14
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: See description	
			Valid value set: 5 -300	
RadioSilence	22	Radio silence timeout	Type: Unsigned32 (MSB)	Units: 1 s See 9.4.2.15
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 600	
			Valid value set: Limited to 1 to 600 for radio silence profile; otherwise 0 – (2 <sup>32</sup> -1)	
RadioSleep	23	Radio sleep period. Note: DLs radio will be disabled when this attribute is set	Type: Unsigned32 (MSB)	Units: 1 s See 9.4.2.17
			Classification: Dynamic	
			Accessibility: Read/write	
			Initial default value: 0	
			Valid value set: 0 – (2 <sup>32</sup> -1)	
RadioTransmitPower	24	Radios maximum transmit power level,	Type: Integer8	Units: dBm See 9.4.2.18
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: See text	
			Valid value set: -20 - +36	
CountryCode	25	Information about the device's regulatory environment	Type: Unsigned16	See 9.4.2.19
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0x0400	
			Valid value set: 4 – 0x3B7E	
Candidates	26	A list of candidate neighbors discovered by the DL	Type: OctetString See Table 151	See 9.4.2.24
			Classification: Dynamic	
			Accessibility: Read/write	
			Initial default value: Null	
			Valid value set: N/A	
DiscoveryAlert	27	Control of NeighborDiscovery alert	Type: OctetString	See 9.4.2.24
			Classification: Dynamic	
			Accessibility: Read/write	
			Initial default value: See 9.4.2.25	
			Valid value set: See 9.4.2.24	
SmoothFactors	28	Smoothing factors for diagnostics	Type: OctetString See Table 153	See 9.4.2.25
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: See Table 153	
			Valid value set: See Table 153	

<b>Standard object type name: dlmo (DL management object)</b>				
<b>Standard object type identifier: 124</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
QueuePriority	29	Queue buffer capacity for specified priority level	Type: OctetString See Table 155 Classification: Static Accessibility: Read/write Initial default value: N=0 Valid value set: N/A	See 9.4.2.26
Ch	30	Channel hopping patterns	Type: OctetString (indexed) See Table 159 Classification: Static Accessibility: Read/write Initial default value: See 9.4.3.2 Valid value set: See 9.4.3.2	See 9.4.3.2
ChMeta	31	Metadata for Ch attribute	Type: Meta_Data_Attribute Classification: Static Accessibility: Read only Initial default value: N/A Valid value set: N/A	
TsTemplate	32	Timeslot templates	Type: OctetString (indexed) See Table 161 and Table 163 Classification: Static Accessibility: Read/write Initial default value: See 9.4.3.3 Valid value set: See 9.4.3.3	See 9.4.3.3
TsTemplateMeta	33	Metadata for TsTemplate attribute	Type: Meta_Data_Attribute Classification: Static Accessibility: Read only Initial default value: N/A Valid value set: N/A	
Neighbor	34	Neighbors	Type: OctetString (indexed) See Table 168 Classification: Static Accessibility: Read/write Initial default value: Empty Valid value set: See 9.4.3.4	See 9.4.3.4
NeighborDiagReset	35	Used to update DiagLevel field within Neighbor attribute	Type: OctetString (indexed) See Table 172 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: See 9.4.3.4.3	See 9.4.3.4.3
NeighborMeta	36	Metadata for Neighbor attribute	Type: Meta_Data_Attribute Classification: Static Accessibility: Read only Initial default value: N/A Valid value set: N/A	
Superframe	37	Superframes; structures and activation	Type: OctetString (indexed) See Table 175 Classification: Dynamic Accessibility: Read/write Initial default value: Empty Valid value set: See 9.4.3.5	See 9.4.3.5
Superframeldle	38	Used to update idle fields within Superframe attribute	Type: OctetString (indexed) See Table 177 Classification: Dynamic Accessibility: Read/write Initial default value: N/A	See 9.4.3.5.3

<b>Standard object type name: dlmo (DL management object)</b>				
<b>Standard object type identifier: 124</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
			Valid value set: N/A	
SuperframeMeta	39	Metadata for Superframe attribute	Type: Meta_Data_Attribute	
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
Graph	40	Graphs	Type: OctetString (indexed) See Table 178	See 9.4.3.6
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Empty	
			Valid value set: N/A	
GraphMeta	41	Metadata for Graph attribute	Type: Meta_Data_Attribute	
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
Link	42	Links	Type: OctetString (indexed) See Table 180	See 9.4.3.7
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Empty	
			Valid value set: N/A	
LinkMeta	43	Metadata for Link attribute	Type: Meta_Data_Attribute	
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
Route	44	Routes	Type: OctetString (indexed) See Table 185	See 9.4.3.8
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Empty	
			Valid value set: N/A	
RouteMeta	45	Metadata for Route attribute	Type: Meta_Data_Attribute	
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
NeighborDiag	46	Neighbor link diagnostics	Type: OctetString (indexed) See Table 187	See 9.4.3.9
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: Empty	
			Valid value set: N/A	
DiagMeta	47	Metadata for NeighborDiag attribute	Type: Meta_Data_Attribute	Counts of octets, not entries. See 9.4.3.9
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
ChannelDiag	48	Per-channel diagnostics for spectrum management	Type: OctetString	See 9.4.2.27
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: See 9.4.2.27	
			Valid value set: N/A	

<b>Standard object type name: dlmo (DL management object)</b>				
<b>Standard object type identifier: 124</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
AlertPolicy	49	Report diagnostics if connectivity problems are detected between regular HRCO reports	Type: OctetString	See 9.6.1
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: See 9.6.1	
			Valid value set: N/A	
DLTimeout	50	DL may reasonably reset to provisioned state if it doesn't receive a time update in this time interval	Type: Unsigned16 (MSB)	See 9.4.2.15
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: See description	
			Valid value set: 1 – 65536	

#### 9.4.2.2 **dlmo.ActScanHostFract**

dlmo.ActScanHostFract configures the device's behavior as an active scanning host, as specified in 9.1.13.4. The setting indicates the fraction of time that the DL should respond when it receives an active scanning solicitation. The default of 0 indicates that the device is not configured as an active scanning host.

#### 9.4.2.3 **dlmo.AdvJoinInfo and dlmo.AdvSuperframe**

dlmo.AdvJoinInfo and dlmo.AdvSuperframe configure the contents of an advertisements DAUX sub-header. Their meaning is described in 9.3.5.2.5.

#### 9.4.2.4 **dlmo.SubnetID**

dlmo.SubnetID is the identifier for the single subnet that the DL is currently using or attempting to join. In this standard, all of the DL management SAPs are designed to handle only one active subnet at a time. If a given device is participating in multiple subnets simultaneously, this may be modeled as multiple instances of the DL. dlmo.SubnetID=0 shall never be used as a subnet ID in this standard, and indicates that the device is not participating in a subnet. dlmo.SubnetID=1 shall be used exclusively to identify provisioning networks. The system manager doesn't set the device's SubnetID directly; rather, it is set by the device itself in the process of discovering and joining the network. See 9.1.10.2.

NOTE In the IEEE Std 802.15.4 design, Subnet ID can be used as a filter for incoming MPDUs. As discussed in 9.1.10.2, a DMIC provides an additional, stronger and more reliable filter once the device has joined the subnet.

#### 9.4.2.5 **dlmo.SolicTemplate**

dlmo.SolicTemplate is a template for the DAUX subheader in a solicitation. When a solicitation is transmitted, the exact data in this OctetString (without the length) shall be used as the DAUX subheader. It is blank (zero length) by default. See 9.3.5.3.

#### 9.4.2.6 **dlmo.TaiTime**

dlmo.TaiTime, when read by the DMAP, is reported as the DL's best estimate of DL time at that instant. See 12.22.4.2 for encoding of TAINetworkTimeValue.

NOTE The dlmo.TaiTime attribute is described as a read-only attribute, where time is acquired by the DL from its neighbors and provided as a service to other layers through the DMAP. This style of specification is not intended to preclude implementations, such as on devices that are clock masters, where time is provided to the DL from an alternative source.

#### 9.4.2.7 **dlmo.MaxBackoffExp**

dlmo. MaxBackoffExp is the maximum backoff exponent for retries; see 9.1.8.2 for a discussion of exponential backoff.

#### **9.4.2.8 dlmo. MaxDsduSize**

dlmo. MaxDsduSize is the maximum number of octets that can be accommodated in a single DSDU. This is used by the network layer to make fragmentation decisions. Its default value of 96 allows for the following constraints:

- A single EUI-64 in the MHR. See 9.3.3.2.
- A one-octet crypto key identifier and a slow hopping time offset in the DMXHR. See 9.3.3.4.
- A single compressed route in DROUT (i.e. no source routing beyond the single hop case). See 9.3.3.6.
- No DAUX, so that a fragmented DSDU cannot be combined with an advertisement, with the exception of the link activation DAUX when 16-bit addressing is used.
- A DMIC-32, not DMIC-64 or DMIC-128.

NOTE MaxDsduSize was calculated as follows: 15 octets for the MHR (see 9.3.3.2); 1 octet for the DHR (see 9.3.3.3); 3 octets for the DMXHR (see 9.3.3.4); 0 octets for the DAUX (see 9.3.3.5); 2 octets for the DROUT (see 9.3.3.6); 4 octets for DADDR (see 9.3.3.7); 4 octets for the DMIC; and 2 octets for the FCS. This total of 31 octets is subtracted from the PSDU capacity of 127 octets, to arrive at a MaxDsduSize of 96 octets.

The system manager shall reduce the value of dlmo.MaxDsduSize as needed if additional constraints apply to a particular configuration.

#### **9.4.2.9 dlmo.MaxLifetime**

dlmo.MaxLifetime the maximum duration, in units of  $\frac{1}{4}$  s, for which a DPDU is be held in the message queue of a single device before it shall be discarded. dlmo.MaxLifetime can be overridden by dlmo.Graph[].MaxLifetime (see 9.4.3.6).

#### **9.4.2.10 dlmo.NackBackoffDur**

dlmo.NackBackoffDur is the duration of the backoff, in units of  $\frac{1}{4}$  s, after receiving a NACK (see 9.1.9.4.4).

#### **9.4.2.11 dlmo.LinkPriorityXmit and dlmo.LinkPriorityRcv**

dlmo. LinkPriorityXmit and dlmo.LinkPriorityRcv are the default priorities to be used when selecting links. If no priority is specified in dlmo.Link[].Priority, use these priorities. For T/R links, use dlmo.LinkPriorityRcv as the priority for the receive side of the link. Link priorities are functionally described in 9.1.8.5.

#### **9.4.2.12 dlmo.IdleChannels**

dlmo.IdleChannels provides a list of channels that shall be idle, as a quick way for the system manager to block the usage of certain channels on a particular device without requiring a coordinated change of channel hopping schedules. A link occurring on any of the channels designated as idle (value 1) by dlmo.ActiveChannels shall be treated as idle. Values of 1 in dlmo.IdleChannels shall not cause hop sequences to be shortened, but rather leaves the hop sequences intact and simply causes all links on designated channels to be treated as idle. (Shortening of the hop sequences themselves is accomplished through a different attribute dlmo.Superframe[].ChMap.) Bit positions 0 to 15 correspond to channels 0 to 15. A bit value of 1 indicates that links using the channel shall be treated as idle. dlmo.IdleChannels is complimentary with dlmo.DeviceCapability.ChannelMap; in operation of the DL, the two are logically combined as follows, resulting in a set of channels that are treated as active by the DL.

((NOT dlmo.IdleChannels) AND (dlmo.DeviceCapability.ChannelMap))

#### **9.4.2.13 dlmo. ClockExpire**

dlmo. ClockExpire is the maximum number of seconds that the DL can safely operate without a clock update. The default is 1000/DeviceCapability.ClockStability, which is intended to keep the DL synchronized to within one millisecond during the join process and thereafter. See 9.1.9.2.2.

#### **9.4.2.14 dlmo.ClockStale**

dlmo.ClockStale determines when the DL should start accepting time updates from secondary DL clock sources.. For example, if dlmo.ClockTimeout is set to the default of 45 seconds, then a DL clock recipient should not accept clock updates from a secondary DL clock source until it has not received a clock update from a primary DL clock source for at least 45 seconds. The default value is 500/DeviceCapability.ClockStability (i.e. half of ClockExpire). See 9.1.9.2.3 for more information.

#### **9.4.2.15 dlmo.DLTimeout**

dlmo.DLTimeout is the maximum number of seconds that the DL can reasonably operate in a DL subnet before resetting itself to the provisioned state. The default value is 2000/DeviceCapability.ClockStability (i.e. twice ClockExpire). See 9.1.9.2.2.

#### **9.4.2.16 dlmo.RadioSilence**

dlmo.RadioSilence designates when a device shall disable its transmitter after losing its network connection. See 9.1.15.4 for more information.

#### **9.4.2.17 dlmo.RadioSleep**

dlmo.RadioSleep is used to disable the DL's radio for a period of time. See 9.1.15.4 for more information. Activation of this attribute shall be slightly delayed to allow for transmitting an application layer acknowledgement of the DMAP TPDU that causes the attribute to be set.

#### **9.4.2.18 dlmo.RadioTransmitPower**

dlmo.RadioTransmitPower is used to control the DL's radio transmit power level, in dBm EIRP. It defaults to the device's maximum supported transmit power level, and is reported during the join process through dlmo.DeviceCapability.RadioTransmitPower. See 9.1.15.5.

#### **9.4.2.19 dlmo.CountryCode**

dlmo.CountryCode provides guidance to a device regarding the regulatory environment. See 9.1.15.6.

#### **9.4.2.20 Subnet filters**

A subnet filter attribute is a string of 4 octets that specifies how a device shall filter incoming advertisements or solicitations. AdvFilter is used to filter incoming advertisements, and SolicFilter is used to filter incoming solicitations.

AdvFilter and SolicFilter each include two fields, a 16-bit BitMask field and a 16-bit TargetID field. Table 142 shows the structure of each subnet filter.

**Table 142 – Subnet filter octets**

octets	bits							
	7	6	5	4	3	2	1	0
2 octets	BitMask (octet 0)							
	BitMask (octet 1)							
2 octets	TargetID (octet 0)							
	TargetID (octet 1)							

Unlike most dlmo attributes, subnet filters use LSB octet ordering conventions. This reflects their use, which is to perform bit comparison operations with DPDUs header elements that are transmitted using LSB.

When a device receives an advertisement, it shall check the incoming DPDUs SubnetID. The advertisement shall be ignored unless:

$$(DPDU.SubnetID \text{ AND } AdvFilter.BitMask) == (AdvFilter.TargetID \text{ AND } AdvFilter.BitMask)$$

AdvFilter.BitMask shall default to 0xffff, and AdvFilter.TargetID shall default to 0x0001, with the result that an unprovisioned device in the default state will filter all advertisements except those received from a provisioning network with SubnetID=1.



#### 9.4.2.23 Data link layer management object device capabilities

The dlmo.DeviceCapability attribute includes various elements that indicate the capabilities of the device. This is a read-only attribute that does not change in normal operation. dlmo.DeviceCapability shall be reported to the system manager as part of the join process.

The OctetString comprises a series of fields that are described in Table 147. Some of these fields, listed as constant, do not change during operation and are reported only on joining. Other fields, listed as dynamic, may change during operation and are also available after joining through identically-named DLMO attributes.

**Table 147 – dlmo.DeviceCapability OctetString fields**

Field name	Field encoding
QueueCapacity (capacity of the queue that is available for forwarding operations)	Type: ExtD卢Int (constant)
ClockStability (nominal clock stability of this device, in ppm)	Type: Unsigned8 (constant)
ChannelMap (map of radio channels supported by the device)	Type: Unsigned16 (constant)
DLRoles (DL roles supported by the device)	Type: Bit8 (constant)
EnergyDesign (copy of attribute dlmo.EnergyDesign)	Type: OctetString
EnergyLeft (copy of attribute dlmo.EnergyLeft)	Type: Integer16
Ack_Turnaround (Time to turnaround an ACK/NACK)	Type: ExtD卢Int (constant)
NeighborDiagCapacity (memory capacity for dlmo.NeighborDiag)	Type: ExtD卢Int (constant)
RadioTransmitPower (copy of attribute RadioTransmitPower)	Type: Integer8
SupportedCCAmodes (bitmap description of CCA modes supported by the device)	Type: Bit8 (constant)
Options (optional features supported by DL)	Type: Bit8 (constant)

Table 148 illustrates the structure of the OctetString. Length of EnergyDesign includes a one-octet length that starts an OctetString.

**Table 148 – dlmo.DeviceCapability OctetString structure**

octets	bits							
	7	6	5	4	3	2	1	0
1-2 octets	QueueCapacity							
1 octet	ClockStability							
2	ChannelMap (MSB)							
1 octet	DLRoles							
6-9 octet	EnergyDesign (OctetString length, plus 5-8 octet content)							
2	EnergyLeft (MSB)							
1-2 octets	AckTurnaround							
1-2 octets	NeighborDiagCapacity							
1 octet	RadioTransmitPower							
1 octet	SupportedCCAmodes							
1 octet	Options							

Fields include:

- dlmo.DeviceCapability.QueueCapacity is the number of buffers in the queue that is available for forwarding operations. This capacity shall not include internal buffers that may be used for messages that flow through the network layer. This value shall be based on the worst case, wherein all DPDUs are of maximum size. In operation, the actual queue capacity may be larger than this reported value. See 9.1.8.5.
- dlmo.DeviceCapability.ClockStability is the nominal clock stability of the device, in ppm, in the absence of a time correction from the DL subnet. See 9.1.9.2.2.
- dlmo.DeviceCapability.ChannelMap is a list of channels that the device can legally support in the device's regulatory domain, where a value of zero indicates that the channel cannot legally be used by this device. Bit positions 0 to 15 correspond to channels 0 to 15. If the DL is configured with links that refer to such blocked channels, the device shall treat those links as idle. See 9.1.7.2.3.

- `dlmo.DeviceCapability.DLRoles` enumerates the DL role profiles supported by the device, where a value of 1 indicates that the device supports the DL role profile. See 9.1.16 for a discussion of DL roles.
  - Bit 0 corresponds to the DL aspects of an I/O role profile.
  - Bit 1 corresponds to the DL aspects of a router role profile.
  - Bit 2 corresponds to the DL aspects of a backbone router role profile.
  - Bit 3 corresponds to the radio silence role profile. See 9.1.15.4.
  - Bits 4-7 are reserved and shall be set to 0.
- `dlmo.DeviceCapability.EnergyDesign` and `EnergyLeft` are described in 9.1.17 and 9.4.2.22.
- `dlmo.DeviceCapability.AckTurnaround` indicates the time needed by the device to process an incoming DPDU and respond with an ACK or NACK, in units of  $2^{-15}$  s. All devices shall be capable of using the default timeslot templates (see Table 165, Table 166, and Table 167).
- `dlmo.DeviceCapability.NeighborDiagCapacity` indicates the capacity, in octets, of the `NeighborDiag` attribute. Only octets shown in Table 187 are included in `NeighborDiagCapacity`. The system manager indirectly creates OctetStrings in `NeighborDiag` by setting `DiagLevel` fields in the `Neighbor` attribute. The system manager should not configure a device to fill `NeighborDiag` in excess of its stated capacity, and a device may fail to accumulate data if the system manager exceeds this stated capacity. A value of 0x7fff indicates that the capacity is sufficient to collect all available diagnostics for each `dlmo.Neighbor`.
- `dlmo.DeviceCapability.RadioTransmitPower` is the device's maximum supported power level, in dBm EIRP. See 9.1.15.5.
- `dlmo.DeviceCapability.SupportedCCAmodes` is a list of CCA modes that the device supports (See 9.1.9.4.3):
  - Bit 0 = 1 indicates that CCA Mode 1 is supported.
  - Bit 1 = 1 indicates that CCA Mode 2 is supported.
  - Bit 2 = 1 indicates that CCA Mode 3 is supported.
  - Bits 3-7 are reserved and shall be set to 0.
- `dlmo.DeviceCapability.Options` indicates optional features that the device supports:
  - Bit 0 = 1 indicates that group codes are supported in `dlmo.Neighbor`.
  - Bit 1 = 1 indicates that graph extensions are supported in `dlmo.Neighbor`.
  - Bit 2 = 1 indicates that the device is capable of receiving duocast or n-cast acknowledgements.
  - Bit 3 = 1 indicates that the device is capable of supporting `dlmo.Superframe.SfType=1`. This may be needed in some regions for regulatory compliance.
  - Bit 4 = 1 indicates that the device is capable of supporting the `dlmo.Graph.Queue` field which, when set to a nonzero value, reserves queue buffer capacity. Queue capacity should not be so reserved unless `dlmo.DeviceCapability.QueueCapacity` exceeds the minimum requirement for a device's role profile (see Table 485).
  - Bits 5-7 are reserved and shall be set to 0.

#### **9.4.2.24 Candidate neighbors**

The `dlmo.Candidates` attribute is used to provide the system manager with a list of candidate neighbors. The DL autonomously populates this attribute as it receives advertisements from a number of candidate neighbors. This attribute is then forwarded to the system manager so that routing decisions can be made. The system manager may reset `dlmo.Candidates.N=0`, thus signaling to the device to clear its history of received advertisements and resume the neighbor discovery process.

The attribute `dlmo.DiscoveryAlert` (Table 149) provides the system manager with control over neighbor discovery and reporting. The system manager sets `dlmo.DiscoveryAlert`, and later receives a copy of the `dlmo.Candidates` attribute through the `dlmo.NeighborDiscovery` alert. Alternatively, the system manager can read the `dlmo.Candidates` attribute on its own schedule, or arrange to report it periodically through the HRCO.

**Table 149 – `dlmo.DiscoveryAlert` fields**

Field name	Field encoding
Descriptor	Type: Alert Report Descriptor (Table 257) Default: Disabled=False Default: Priority=0
Duration	Type: Unsigned16 Units: 1 s Default: 60

Table 150 illustrates the structure of `DiscoveryAlert`.

**Table 150 – `dlmo.DiscoveryAlert` structure**

octets	Bits							
	7	6	5	4	3	2	1	0
2 octets	Descriptor							
2 octets	Duration							

When `dlmo.DiscoveryAlert` is enabled (Descriptor.Disabled=False), the DL shall report the contents of `dlmo.Candidates`.Duration seconds later using the `dlmo.NeighborDiscovery` alert. Once the DL completes the alert, the DL resets `dlmo.DiscoveryAlert` to zero.

When the `NeighborDiscovery` alert is triggered by the state of Duration (`dlmo.DiscoveryAlert.Duration`), the DL shall automatically set Duration to zero, thereby resulting in a single `NeighborDiscovery` alert each time Duration is set to a nonzero value.

`dlmo.DiscoveryAlert` shall be enabled and default to Duration=60 when the DL enters the joined state. This default indicates that the DL shall, when it enters the joined state, accumulate information from advertisements in `dlmo.Candidates` for a period of 60 s, and then report the results using the `dlmo.NeighborDiscovery` alert. See 9.1.14.6. The system manager may override this default by writing to `dlmo.DiscoveryAlert` in conjunction with the join response.

Advertisements from neighboring routers include links that can be used to communicate with that router. When `DiscoveryAlert` is enabled, the DL may use these links to interrogate, with a zero length DSDU, each candidate router, on multiple channels, and receive signal quality metric in the DAUX. This information enables the DL to build a more accurate picture of signal quality in `dlmo.Candidates`.

When `dlmo.DiscoveryAlert` is disabled, the DL should nonetheless passively build a `dlmo.Candidates` list as a background process after it joins the DL subnet, based on whatever advertisements it happens to receive in the course of operating the DL state machine.

If there is a `dlmo.DL_Connectivity` alert and `DiscoveryAlert` is enabled, the DL shall also send a `dlmo.NeighborDiscovery` alert at the same time.

When `dlmo.DiscoveryAlert.Duration` is set to 0, the device shall not send `dlmo.NeighborDiscovery` alerts on a timed basis. The device should continue to maintain the `Candidates` attribute so that it can be read as needed by the system manager by reading the `dlmo.Candidates` attribute directly or by configuring the device to report it periodically through the HRCO.

The process of scanning for advertisements is described in 9.1.13.

dlmo.Candidates comprises a series of fields that are described in Table 151. In essence, the <Candidate>,<RSQI> tuple is repeated N times, providing an indication of signal quality to multiple neighbors.

dlmo.Candidates may reasonably exclude current entries in the dlmo.Neighbor table, when the same information for those neighbors is available through the neighbor diagnostics.

**Table 151 – dlmo.Candidates OctetString fields**

Field name	Field encoding
N (count of discovered neighbors)	Type: Unsigned8
Neighbor <sub>1</sub> (16-bit address of first candidate)	Type: ExtDUIint
RSSI <sub>1</sub> (radio signal quality of first candidate)	Type: Integer8
RSQI <sub>1</sub> (radio signal quality of first candidate)	Type: Unsigned8
etc...	
Neighbor <sub>N</sub> (16-bit address of Nth candidate)	Type: ExtDUIint
RSSI <sub>N</sub> (radio signal quality of Nth candidate)	Type: Integer8
RSQI <sub>N</sub> (radio signal quality of Nth candidate)	Type: Unsigned8

Table 152 illustrates the structure of Candidates.

**Table 152 – dlmo.Candidates structure**

octets	bits								
	7	6	5	4	3	2	1	0	
1 octet	N								
0-2 octets	Neighbor <sub>1</sub>								
0-1 octet	RSSI <sub>1</sub>								
0-1 octet	RSQI <sub>1</sub>								
	etc....								
0-2 octets	Neighbor <sub>N</sub>								
0-1 octet	RSSI <sub>N</sub>								
0-1 octet	RSQI <sub>N</sub>								

Fields include:

- dlmo.Candidates.N is the number of neighbors that have been discovered. A device shall support at least 5 candidate entries. If many neighbors are discovered, the DL may report only the best candidates based on the quality of the radio link.
- dlmo.Candidates.Neighbor<sub>N</sub> is the 16-bit address of each candidate neighbor in the DL subnet.
- dlmo.RSSI<sub>N</sub> indicates the strength of the radio signal from each candidate neighbor, based on received advertisements and possibly other DPDUs. See 9.1.15.2 for description of RSSI.
- dlmo.RSQI<sub>N</sub> indicates the quality of the radio signal from each candidate neighbor, based on received advertisements and possibly other considerations. A higher number indicates a better radio signal. See 9.1.15.2. for description of RSQI. If the chipset does not support RSQI, i.e., if RSQI=0, then the DL shall map available RSSI into RSQI units in this context.

#### 9.4.2.25 Smoothing factors

The dlmo.SmoothFactors provides the smoothing factors for the dlmo.NeighborDiag attribute. The use of these factors is described in 9.1.15.3.

Three fields in dlmo.NeighborDiag involve exponential smoothing: RSSI, RSQI, and ClockBias. The smoothing factor  $\alpha$  (alpha) for each of these values is individually configurable. Alpha is expressed as an integer percentage in the range of 0 to 100. RSSI and

RSQI default to 10%, so the values tend to reflect recent history. ClockBias defaults to 1%, since that value is intended to show clock bias over an extended period of time.

Each smoothing factor involves three values: xxxAlphaHigh, xxxAlphaLow, and xxxThreshold. If the new data is below the threshold, use xxxAlphaLow as the smoothing factor; otherwise use xxxAlphaHigh.

The fields in dlmo.SmoothFactors are described in Table 153.

**Table 153 – dlmo.SmoothFactors OctetString fields**

Field name	Field encoding	Default
RSSI_Threshold (Threshold for RSSI)	Type: Integer16	0
RSSI_AlphaLow (AlphaLow for RSSI)	Type: Unsigned8	10
RSSI_AlphaHigh (AlphaHigh for RSSI)	Type: Unsigned8	10
RSQI_Threshold (Threshold for RSQI)	Type: Integer16	0
RSQI_AlphaLow (AlphaLow for RSQI)	Type: Unsigned8	10
RSQI_AlphaHigh (AlphaHigh for RSQI)	Type: Unsigned8	10
ClockBias_Threshold (Threshold for ClockBias)	Type: Integer16	0
ClockBias_AlphaLow (AlphaLow for ClockBias)	Type: Unsigned8	1
ClockBias_AlphaHigh (AlphaHigh for ClockBias)	Type: Unsigned8	1

Table 154 illustrates the structure of SmoothFactors.

**Table 154 – dlmo.SmoothFactors structure**

octets	bits								
	7	6	5	4	3	2	1	0	
2 octets	RSSI_Threshold (MSB)								
1 octet	RSSI_AlphaLow								
1 octet	RSSI_AlphaHigh								
2 octets	RSQI_Threshold (MSB)								
1 octet	RSQI_AlphaLow								
1 octet	RSQI_AlphaHigh								
2 octets	ClockBias_Threshold (MSB)								
1 octet	ClockBias_AlphaLow								
1 octet	ClockBias_AlphaHigh								

#### 9.4.2.26 dlmo.QueuePriority

##### 9.4.2.26.1 General

dlmo.QueuePriority is an attribute that enables the system manager to specify the nominal buffer capacity in the DL's forwarding queue for specific priority levels. As described in 9.1.8.5, the system manager may configure a device's nominal buffer capacity to limit the number of buffers that can be used to forward low priority DPDUs. For example, the system manager may configure dlmo.QueuePriority so that no more than 3 buffers shall be used to forward DPDUs with priority <=2.

The nominal capacity of the forwarding queue can be found in dlmo.DeviceCapability.QueueCapacity (see 9.4.2.23).

##### 9.4.2.26.2 Semantics

Table 155 specifies the fields for dlmo.QueuePriority.

**Table 155 – dlmo.QueuePriority fields**

Field name	Field encoding
N (count of priorities specified, 0 to 15, default N=0)	Type Unsigned8
Priority <sub>1</sub> (first priority)	Type: Unsigned8
QMax <sub>1</sub> (first buffer capacity)	Type: Unsigned8
etc...	
Priority <sub>N</sub> (Nth priority)	Type: Unsigned8
QMax <sub>N</sub> (Nth buffer capacity)	Type: Unsigned8

For example, if Priority is 2 and QMax is 3, then no more than 3 queue buffers shall be used to forward DPDUs with priority <= 2. This count shall not include DPDUs that are using queue capacity that was set aside for DPDUs being forwarded along a particular graph, based on dlmo.Graph[].Queue. Priority shall be enumerated in increasing order, so that Priority<sub>X</sub> shall be less than Priority<sub>X+1</sub>. Similarly, QMax<sub>X</sub> shall be less than QMax<sub>X+1</sub>.

The count QMax<sub>X</sub> sets the maximum available slots available for low-priority messages, not reserved slots for low-priority messages. In effect, QMax<sub>X</sub> ensures that the remainder of the queue is available for DPDUs of priority 1+PriorityX.

As described in 9.2.2, the DE indicator in the DPDUs header indicates whether a DPDUs on the queue is eligible to be discarded in favor of an incoming DPDUs of higher priority.

The default, where N=0, indicates that the system manager has not configured a limit on the number of forwarding buffers that can be used for low priority DPDUs.

Table 156 illustrates the structure of dlmo.QueuePriority.

**Table 156 – dlmo.QueuePriority structure**

octets	bits							
	7	6	5	4	3	2	1	0
1	N							
2 (opt)	Priority <sub>1</sub>							
3 (opt)	QMax <sub>1</sub>							
etc...								
2N (opt)	Priority <sub>N</sub>							
2N+1 (opt)	QMax <sub>N</sub>							

#### 9.4.2.27 dlmo.ChannelDiag

##### 9.4.2.27.1 General

dlmo.ChannelDiag is a read-only dynamic attribute that reports DPDUs transmit failure rates on each radio channel supported by the standard. This enables the system manager to be aware of consistent connectivity problems on a per-channel basis, as a diagnostic for spectrum management in a subnet.

Two diagnostics are reported for each channel, each indicating a different type of failure. NoACK<sub>N</sub> indicates the percentage of time that unicast DPDUs transmissions on channel N did not receive an ACK or a NACK. CCABackoff<sub>N</sub> indicates the percentage of time that unicast DPDUs transmissions on channel N were aborted due to CCA.

Under some situations, CCA backoff is part of normal DL subnet operation and is not indicative of poor channel quality. In particular, contentions for shared links will cause CCA backoffs. Therefore, the DL may be selective in counting CCA backoffs. It is recommended that CCA backoffs should not normally be counted if they occur in shared links within slotted hopping superframes. However, for shared links within slot hopping superframes, CCA backoffs may reasonably be counted. Shared links are described in 9.1.8.2

ChannelDiag values are 8-bit signed integers.

- A value of 0 indicates that no unicast transmission has been attempted on the channel.
- Positive values in the range of 1 to 101 indicate the percentage failure rate plus one. For example, a CCABackoff<sub>6</sub> value of 26 indicates that 25% of unicast transmissions on channel 6 were aborted due to CCA.
- Negative values in the range of -1 to -101 indicate the percentage failure rate as a negative number minus one. Failure rates are reported as negative numbers if they are based on 5 or fewer attempted transmissions. For example, a NoACK<sub>8</sub> value of -34 indicates that 33% of unicast transmissions on a particular channel did not receive an acknowledgement, and that 5 or fewer transmissions have been attempted on that channel.

The DL may selectively skip transmission links in anticipation of a failed transmission, as described in 9.1.7.2.4. Such skipped links should be treated as equivalent to NACK for the applicable channel.

Each time ChannelDiag is read by the system manager or reported periodically through the HRCO, its underlying accumulators shall be reset to zero.

#### 9.4.2.27.2 Semantics

Table 157 specifies the fields for dlmo.ChannelDiag.

**Table 157 – dlmo.ChannelDiag fields**

Field name	Field encoding
Count (Number of attempted unicast transmissions for all channels)	Type: Unsigned16
NoACK <sub>0</sub> (Percentage of time transmissions on channel 0 did not receive an ACK)	Type: Integer8
CCABackoff <sub>0</sub> (Percentage of time transmissions on channel 0 aborted due to CCA)	Type: Integer8
etc...	
NoACK <sub>15</sub> (Percentage of time transmissions on channel 15 did not receive an ACK)	Type: Integer8
CCABackoff <sub>15</sub> (Percentage of time transmissions on channel 15 aborted due to CCA)	Type: Integer8

Table 158 illustrates the structure of dlmo.ChannelDiag.

**Table 158 – dlmo.ChannelDiag structure**

octet	bits							
	7	6	5	4	3	2	1	0
1	Count (MSB)							
2								
3	NoACK <sub>0</sub>							
4	CCABackoff <sub>0</sub>							
etc...								
33	NoACK <sub>15</sub>							
34	CCABackoff <sub>15</sub>							

#### 9.4.3 Data link layer management object attributes (indexed OctetStrings)

##### 9.4.3.1 General

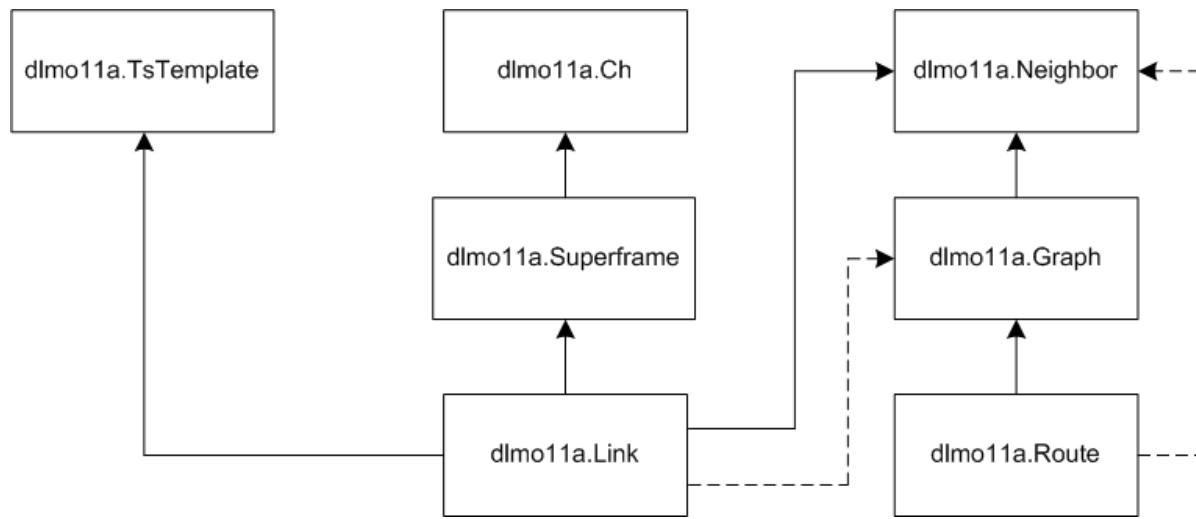
Indexed OctetString management object attributes are structured collections of data, similar in concept to SQL tables. For example, the DL maintains a list of neighbors in an indexed OctetString attribute called dlmo.Neighbor, where each neighbor can be visualized as a row in a table, with the structure of each row as shown in Table 168 and Table 169. A 15-bit index for each row, in this case corresponding to the 16-bit address of the neighbor, is provided for each indexed OctetString attribute in the DLMO.

For consistency of processing, all indexed OctetString attributes in the DL include an index in the first field that is type ExtDUIint. However, in the case of dlmo.Ch, dlmo.TsTemplate, and

dlmo.Superframe, the index is limited to a range of 1 to 127, thus guaranteeing that the index can be represented in a single octet. References to these structures can also be compressed to a single octet.

Indexed OctetString index of zero shall not be used in the DLMO, except to indicate a null entry.

Figure 90 illustrates the relationship among some of the indexed OctetString DLMO attributes. Referential relationships are shown with arrows. For example, dlmo.Link refers to dlmo.TsTemplate, so an arrow is shown pointing in the direction of the reference. This roughly corresponds to a one-to-many relationship, where one template can be referenced by multiple links.



**Figure 90 – Relationship among DLMO indexed attributes**

As shown in Figure 90, dlmo.TsTemplate is referenced by dlmo.Link. Timeslot templates specify transaction structure and timing when a link is used.

dlmo.Link, dlmo.Superframe, and dlmo.Ch are related. Superframes select a channel hopping pattern and a cyclic schedule. Links describe the various actions that are taken during each superframe cycle. Each link refers to one superframe.

For unicast (and duocast) transactions, dlmo.Link refers to dlmo.Neighbor. A single reference may encompass a group of neighbors. If a link is reserved for use of a certain graph, or gives preferential access to a certain graph, then dlmo.Link will also refer to dlmo.Graph. Since it is possible to configure a link without a reference to a graph, this reference is shown as a dotted line in Figure 90.

dlmo.Route and dlmo.Graph are related in that routes typically refer to graphs. dlmo.Graph and dlmo.Neighbor are related in that graphs refer to neighbors. dlmo.Route and dlmo.Neighbor are loosely related (indicated by a dotted line in Figure 90) in that source routing may implicitly refer to a neighbor.

An additional relationship, not shown in Figure 90, exists between dlmo.Neighbor and dlmo.NeighborDiag. Both are indexed by the 16-bit address of the device's DL neighbors, but with different purposes.

- dlmo.Neighbor: The system manager provides this static indexed OctetString attribute to enable the DL to communicate with its immediate neighbors.
- dlmo.NeighborDiag: The system manager configures this dynamic indexed OctetString attribute to enable the DL to collect and periodically report diagnostics for its immediate neighbors.

### 9.4.3.2 dlmo.Ch

#### 9.4.3.2.1 General

dlmo.Ch is an indexed OctetString collection that contains available channel hopping patterns.

Channel hopping patterns 1 through 5 are reserved as standard defaults, as described in 9.1.7.2.5. Additional hopping patterns may be added.

Each device can store multiple channel hopping patterns, with a unique index for each pattern. Advertisements assume that hop patterns for the join process are configured in the device when the advertisement is received, so that any hop pattern referenced in an advertisement shall match one of the defaults.

The system manager inserts, updates, or deletes channel hopping patterns by sending the DMAP a hopping pattern, along with a unique index and (if selected) a TAI cutover time.

For a given hopping pattern, the standard provides a mapping between the DL channel numbers and the more flexible MAC channel numbers. As applied to the 2.4 GHz DSSS IEEE Std 802.15.4 radio, channel numbers shall be limited to the range of 0 to 15, corresponding to IEEE Std 802.15.4 channel numbers 11 to 26 (channel page 0), in the same order. Hop patterns for this radio shall not exceed a length of 16.

**NOTE** dlmo.Ch data types are limited to radios with 16 or fewer channels. Future radios with more channels would involve providing support for a less compressed representation.

Five default hop patterns are reserved and defined by the standard. Default hop patterns are enumerated and described in 9.1.7.2.5.

#### 9.4.3.2.2 Semantics

Table 159 specifies the fields for dlmo.Ch. An index, used to identify each channel hopping pattern, is consistent with DL conventions for indexed OctetStrings.

**Table 159 – dlmo.Ch fields**

Field name	Field encoding
* Index	Type: ExtDIUInt (used as an index) Valid value set: 1 - 127
Length	Type: Unsigned8 Valid value set: 1 - 16
Seq (Channel hopping pattern, with Length entries)	Type: SEQUENCE OF Unsigned4 (SIZE (Length))

Table 160 illustrates the structure of dlmo.Ch.

**Table 160 – dlmo.Ch structure**

octets	bits								
	7	6	5	4	3	2	1	0	
1 octet (range 1-127)	Index								
1 octet	Length								
(Length+1)/2 octets	Seq <sub>1</sub>				Seq <sub>0</sub>				
	Seq <sub>3</sub>				Seq <sub>2</sub>				
	...				...				
	0000 if Length is an odd number Seq <sub>Length-1</sub> if Length is an even number				Seq <sub>2(n-2)</sub>				

### 9.4.3.3 dlmo.TsTemplate

#### 9.4.3.3.1 General

dlmo.TsTemplate is an indexed OctetString collection that contains timeslot templates. Timeslot templates describe DL transaction timing.

Timeslot templates 1, 2, and 3 shall be reserved as standard defaults, as enumerated in Table 165, Table 166, and Table 167. Additional timeslot templates may be added by the system manager.

The system manager inserts, updates, or deletes timeslot templates by sending the DMAP a template, along with a unique index and (if selected) a TAI cutover time.

Template time offsets are specified in units of  $2^{-20}$  s, which is approximately 1  $\mu$ s. Timeslot length is set by the superframes that use the template.

Template types include receive and transmit templates. Both template types include acknowledgements for unicast transactions. The same templates can also be used for broadcast links, such as solicitations, that don't need acknowledgements and don't use the acknowledgement timings.

Templates can be defined on three levels:

- Default templates are defined in the standard. These are the timeslot templates needed for joining (Table 165, Table 166, and Table 167). Template index=1 describes a generic receive transaction, and is used for receiving join responses. Template index=2 describes a generic transmit transaction, and is used for transmitting join requests. Template index=3 describes a transaction that operates its receiver for an entire timeslot, intended for operations such as scanning for advertisements or receiving loosely timed slow hopping DPDUs. These generic templates shall be used during provisioning and joining, and may also be used for other purposes.
- Provisioned templates may be added during the provisioning process, with a lifetime that lasts until the DL has joined the network. See 9.1.14.4.
- Subnet-specific templates may be provided to the DL after the join process is completed.

DPDU transmission timing is based on the device's internal clock. Acknowledgement timing is specified as a time offset from a reference point that is indicated in the template. Typically, the configured time range for a device transmitting a DPDUs is narrower than the time range for the receiving device, to account for guard times.

Timeslot templates shall always provide a reception window of at least 192  $\mu$ s, which implies that devices shall be capable of controlling transmission timing with an accuracy of at least  $\pm 96 \mu$ s, i.e.,  $\pm 6$  PHY symbol periods.

By convention, timeslot template timing is specified based on the start and end times of DPDUs. PPDU timing, dependent on the details of the physical layer that contains the DPDUs, can be inferred from PPDU times. PPDU start time, as specified in timeslot templates, uses a convention that the PPDU begins at the instant just before the first octet in the PPDU header is transmitted. This convention applies to PPDU transmissions as well as acknowledgements.

**NOTE** In actual implementations based on IEEE Std 802.15.4 (2.4 GHz), PhL timing signals may be triggered when an SFD (start frame delimiter) is completely transmitted or received. The start time of the PPDU is 1 octet, or 32  $\mu$ s, after that reference point.

ACK/NACK response time is commonly specified in relation to the PPDU end time, with the acknowledgement occurring approximately 1 ms thereafter. Alternatively, acknowledgements may be timed in relation to the timeslot end time. By placing acknowledgements at a fixed offset within the timeslot, it is possible to support routers that operate on multiple frequencies at the same time, sharing a common antenna, or devices in close proximity to each other, where an outbound acknowledgement on one channel might unintentionally jam an incoming PPDU on another channel, or vice versa.

When acknowledgement times are defined as offsets from the end of the timeslot, the time offsets, which are unsigned integers, shall be interpreted as referring to a time prior to the end of the timeslot.

Receive templates specify:

- The time range when the first octet of a DPDU can be received, indicating a time range to enable and disable the radio's receiver. A time range that exceeds the timeslots duration indicates that the range extends to the end of the timeslot.
- Acknowledgement delay time range. A receiving device of a unicast DPDU should respond with an ACK/NACK as early as possible within this range, with the exception of intentionally delayed n-cast acknowledgements. Acknowledgement delay time may be specified in reference to the end of the incoming DPDU or as an offset from the timeslot end time.
- Whether it is acceptable to operate past the timeslot boundary once reception of a DPDU begins, essentially extending timeslot duration. Overrun of a slot boundary can be used to accommodate various cases. For example, slow-hopping devices do not necessarily track slot boundaries accurately, and support of such devices involves allowing reception even when a DPDU overruns a slot boundary.

Transmit templates specify:

- Time range to begin transmission. A compliant device may begin its transmission at any time within the time range, based on its internal DL clock. The time range is based on the start time of the DPDU. The CCA operation, if any, and the PPDU's SHR and PHR precede that event and therefore may occur prior to the time range.
- Acknowledgement delay time range. A receiving device should respond as early as possible within this range. Acknowledgement delay time may be specified in relation to the end of the transmitted DPDU or the timeslot end time.
- Indication of whether and how to check CCA before transmission.
- Indication of whether the DL should periodically continue operating its receiver for the entire acknowledgement delay time range, even after an acknowledgement is received (intended for duocast support; see 9.1.9.4.7).

#### 9.4.3.3.2 Semantics

There are two variations of dmo.TsTemplate, one for a receive template and another for a transmit template. The variations are distinguished by a 2-bit Type that is the first element. Type=0 indicates a receive template, and type=1 indicates a transmit template. Types 2 and 3 are reserved for future use.

Table 161 specifies the fields for the receive template.

**Table 161 – Receive template fields**

<b>Field name</b>	<b>Field encoding</b>
* Index	Type: ExtDUIuint (used as an index) Valid value set: 1 - 127
Type (indicates that this is a receive template)	Type: Unsigned2 0= receive template 1= see Table 163 2 and 3 are reserved
AckRef (indicates reference for acknowledgement time range)	Type: Unsigned1 Valid value set: 0 or 1 0=offset from end of incoming DPDU 1=negative offset from end of timeslot
RespectBoundary (specifies whether or not slot boundaries shall be respected (i.e., whether reception and its acknowledgement may extend past a slot boundary))	Type: Unsigned1 0= slot boundaries do not need to be respected 1= slot boundaries shall be respected
Reserved (octet alignment)	Type: Unsigned4=0
WakeupDPDU (earliest time when DPDU reception can be expected to begin, indicating when to enable radio's receiver; offset from timeslot's scheduled start time)	Type: Unsigned16 Size = 2 octets Units: $2^{-20}$ s
TimeoutDPDU (latest time when DPDU reception can be expected to begin, indicating when to disable receiver if no incoming DPDU is detected; offset from timeslot's scheduled start time)	Type: Unsigned16 Size = 2 octets Units: $2^{-20}$ s
AckEarliest (start of acknowledgement delay time range, with start of ACK's PSDU (MPDU) being the time reference. Semantics depend on AckRef: if AckRef=1, subtract value from scheduled end time of timeslot to determine AckEarliest)	Type: Unsigned16 Size = 2 octets Units: $2^{-20}$ s
AckLatest (end of acknowledgement delay time range, with start of ACK's PSDU (MPDU) being the time reference. Semantics depend on AckRef: if AckRef=1, subtract value from scheduled end time of timeslot to determine AckLatest)	Type: Unsigned16 Size = 2 octets Units: $2^{-20}$ s

Table 162 specifies the receive template structure.

**Table 162 – Receive template structure**

<b>octets</b>	<b>bits</b>													
	7	6	5	4	3	2	1	0						
1 octet	* Index (range 1-127)													
1 octet	Type=0	AckRef	RespectBoundary	Reserved=0										
2 octets	WakeupDPDU (MSB)													
2 octets	TimeoutDPDU (MSB)													
2 octets	AckEarliest (MSB)													
2 octets	AckLatest (MSB)													

Table 163 specifies the fields for the transmit template.

**Table 163 – Transmit template fields**

Field name	Field encoding
* Index	Type: ExtDlUInt (used as an index) Valid value set: 1 - 127
Type (indicates that this is a transmit template)	Type: Unsigned2 0= see Table 161 1= transmit template 2 and 3 are reserved
AckRef (indicates reference for acknowledgement delay time range)	Type: Unsigned1 Valid value set: 0 to 1 0 = offset from end of transmitted DPDU 1= negative offset from end of timeslot
CCAmode (indicates whether to check CCA before transmission)	Type: Unsigned 2 Valid value set: 0 to 3 0= no CCA 1-3=CCA mode as per IEEE standard See 9.1.9.4.3
KeepListening (indicates whether the DL should periodically continue operating its receiver for the full ACK time range, even after an ACK is received; see 9.1.9.4.7)	Type: Unsigned1 Valid value set: 0 to 1 0= no 1= yes
Reserved (octet alignment)	Type: Unsigned2=0
XmitEarliest (earliest time to start DPDU transmission; offset from timeslot's scheduled start time)	Type: Unsigned16 Size = 2 octets Valid value set: $2^{-20}$ s
XmitLatest (latest time to start DPDU transmission; offset from timeslot's scheduled start time)	Type: Unsigned16 Size = 2 octets Valid value set: $2^{-20}$ s
WakeupAck (earliest time when ACK MPDU reception can be expected to begin; enable receiver early enough to receive an ACK MPDU beginning at this time. Semantics depend on ACKref; if AckRef=1, subtract value from scheduled end of timeslot to determine WakeupAck.)	Type: Unsigned16 Size = 2 octets Valid value set: $2^{-20}$ s
TimeoutAck (latest time when ACK reception can be expected to begin. DL may disable receiver if ACK MPDU reception has not started by this time. Semantics depend on ACKref; if AckRef=1, subtract value from scheduled end of timeslot to determine TimeoutAck.)	Type: Unsigned16 Size = 2 octets Valid value set: $2^{-20}$ s

NOTE An AckEarliest value of 402 (384  $\mu$ s) accommodates the IEEE Std 802.15.4 SIFS requirement of 6 octets, plus 6 octets for a PPDU's SHR and PHR prior to the start of the ACK's PSDU (MPDU).

Table 164 specifies the transmit template structure.

**Table 164 – Transmit template structure**

octets	bits												
	7	6	5	4	3	2	1	0					
1 octet	* Index (range 1-127)												
1 octet	Type	AckRef	CheckCCAmode			KeepListening	Reserved=0						
2 octets	XmitEarliest (MSB)												
2 octets	XmitLatest (MSB)												
2 octets	WakeupAck (MSB)												
2 octets	TimeoutAck (MSB)												

Table 165, Table 166, and Table 167 specify the values for the three default DL timeslot templates. These read-only timeslot templates use indexes 1, 2, and 3, and shall be used for links that are specified in advertisements. Their structure is general purpose, and they may be referenced by other links as well.

The DL shall be capable of transmitting and receiving acknowledgements in the  $1032 \pm 100 \mu$ s timing specified in these default templates, or more slowly if so specified in an alternative

timeslot template. The attribute `dlmo.DeviceCapability.AckTurnaround` informs the system manager if a device is capable of handling acknowledgements more quickly.

**Table 165 – Default receive template**

Field	Default value	In microseconds (informative)
* Index	1	
Type	0	
AckRef	0	
RespectBoundary	1	
WakeupDPDU	1271	1212 $\mu$ s
TimeoutDPDU	3578	3412 $\mu$ s
AckEarliest	977	932 $\mu$ s
AckLatest	1187	1132 $\mu$ s

**Table 166 – Default transmit template**

Field	Default value	In microseconds (informative)
* Index	2	
Type	1	
AckRef	0	
CCAmode	1	
KeepListening	0	
XmitEarliest	2319	2212 $\mu$ s
XmitLatest	2529	2412 $\mu$ s
WakeupAck	977	932 $\mu$ s
TimeoutAck	1187	1132 $\mu$ s

**Table 167 – Default receive template for scanning**

Field	Default value	Explanation
* Index	3	
Type	0	
AckRef	0	
RespectBoundary	0	If DPDUs reception commences within the timeslot boundaries, complete processing of transaction
WakeupDPDU	0	Start of timeslot; allowing for timeslots that are contiguous. In the first timeslot of a contiguous series, a device may insert setup time at the start of the first timeslot not to exceed 1271
TimeoutDPDU	0xffff	End of timeslot
AckEarliest	977	Same as default receive template
AckLatest	1187	Same as default receive template

#### 9.4.3.3.3 Default template timings (INFORMATIVE)

Default timeslot templates are intended to match WirelessHART timings.

The default DPDUs transmission time is based on an offset of 2312  $\mu$ s from the start of the timeslot to the start of the DPDUs. The default transmit template accounts for  $\pm 100 \mu$ s of transmitter jitter, resulting in a default range of  $2312 \pm 100 \mu$ s. The default receive template accounts for the same  $\pm 100 \mu$ s of transmit jitter, plus clock drift of  $\pm 1000 \mu$ s, resulting in a receive DPDUs range of  $2312 \pm 1100 \mu$ s.

The default ACK transmission time is based on an offset of 1032  $\mu$ s from the end of the DPDUs to the start of the ACK's MPDUs. The default ACK templates allow for  $\pm 100 \mu$ s of transmit jitter, for a default template of  $1032 \pm 100 \mu$ s. Clock drift is considered inconsequential in the short time span from the end of a DPDUs to the start of an acknowledgement.

In the IEEE Std 802.15.4 radio used in this standard, the physical layer header is 6 octets, or 192  $\mu$ s, in duration. Thus, radio transmission or reception begins 192  $\mu$ s earlier than the times specified in the templates. CCA, if required, precedes the radio startup.

Templates do not account for the internals of radio operation, leaving that as an internal device matter. For example, the value of 932  $\mu$ s for WakeupACK means that the physical layer header is allowed to begin transmission 192  $\mu$ s sooner, or as early as  $932-192=740$   $\mu$ s following the end of the DPDU. The receiver of the acknowledgement, also with a nominal WakeupAck of 932  $\mu$ s, therefore needs its radio to be running at 740  $\mu$ s following the end of the DPDU and ready to start receiving the acknowledgement at that time. If the receiver needs additional time to account for radio startup and receiver jitter, then the radio needs to be enabled sufficiently in advance of 740  $\mu$ s to ensure that it can receive the full physical layer header.

#### **9.4.3.4 dlmo.Neighbor**

##### **9.4.3.4.1 General**

dlmo.Neighbor is an indexed OctetString collection that contains information about immediate unicast neighbors. dlmo.Neighbor entries are referenced by graphs and links.

The system manager inserts, updates, or deletes dlmo.Neighbor entries by sending the DMAP neighbor entries, along with a unique index and (if selected) a TAI cutover time. The neighbor's 16-bit address is used as an index.

The neighbors in the dlmo.Neighbor attribute are set by the system manager, not by the device itself. The DL autonomously builds a list of candidate neighbors in the dlmo.Candidates attribute, as described in 9.4.2.24. This list is then forwarded to the system manager. The system manager considers the radio connectivity that is reported in dlmo.Candidates, but may also consider other criteria such as resource constraints, historical performance, or subnet topology.

When the DL processes an advertisement during the join procedure, the DL automatically adds the advertising router as an entry in the dlmo.Neighbor table, thereby enabling communication through the proxy. This entry persists after the device successfully joins the network, unless it is deleted or updated by the system manager. The system manager infers the existence and content of this entry based on the identity of the proxy that the device uses to join the network. See 9.3.5.2.1.

This standard follows the IETF RFC 4944 convention, whereby 16-bit unicast addresses are limited to the range of 0xxxxxxxxxxxxx. See 9.3.3.6.

Diagnostic information related to neighbors can be found in the attribute dlmo.NeighborDiag (see 9.4.3.9).

##### **9.4.3.4.2 Semantics**

Table 168 specifies the fields for dlmo.Neighbor.

**Table 168 – dlmo.Neighbor fields**

Field name	Field encoding
* Index (16-bit address of neighbor)	Type: ExtDIUInt (used as an index) Valid value set: 1 to 32767
EUI64 (EUI-64 identifier of the neighbor)	Type: Unsigned64
GroupCode1 (associate a group code with a set of neighbors; used by dlmo.Link)	Type: Unsigned8
GroupCode2 (associate a group code with a set of neighbors; used by dlmo.Link)	Type: Unsigned8
ClockSource (indicates whether neighbor is a DL clock source)	Type: Unsigned2 Valid value set: 0 to 2 0 = no 1 = secondary 2 = preferred 3 = reserved
ExtGrCnt (count of graphs virtually extended for this neighbor)	Type: Unsigned2
DiagLevel (selection of neighbor diagnostics to collect)	Type: Unsigned2 Bit0 = 1 Collect link diagnostics Bit1 = 1 Collect clock diagnostics
LinkBacklog (indicates that link information is provided to facilitate clearing message queue backlog to the neighbor)	Type: Bit1 Valid value set: 1=Extra link information is provided 0=No extra link information
Reserved (octet alignment)	Type: Bit1=0
ExtendGraph	Type: SEQUENCE OF Octet2 (SIZE (ExtGrCnt)) Valid value set: See Table 170
LinkBacklogIndex (Activate this link to clear queue backlog)	Type: ExtDIUInt Valid value set: 1-127 Null and not transmitted if LinkBacklog=0 Link index if LinkBacklog=1
LinkBacklogDur (duration of link activation)	Type: Unsigned8 Units: Link occurrences Valid value set: Null and not transmitted if LinkBacklog=0 1 to 255 if LinkBacklog=1
LinkBacklogActivate (link activation criterion)	Type: Unsigned8 Units: DPDUs on queue Valid value set: Null and not transmitted if LinkBacklog=0 1 to 255 if LinkBacklog=1

Table 169 illustrates the structure of dlmo.Neighbor.

**Table 169 – dlmo.Neighbor structure**

octets	bits											
	7	6	5	4	3	2	1	0				
1-2 octets	* Index											
8 octets	EUI64											
1 octet	GroupCode1											
1 octet	GroupCode2											
1 octet	ClockSource	ExtGrCnt	DiagLevel		LinkBacklog	Reserved						
2 * ExtGrCnt octets	ExtendGraph <sub>1</sub> ... ExtendGraph <sub>ExtGrCnt</sub>											
0 - 1 octet	LinkBacklogIndex											
0 - 1 octet	LinkBacklogDur											
0 - 1 octet	LinkBacklogActivate											

Fields include:

- EUI64. In order to communicate with a neighbor, the security sub-layer needs the EUI-64 of that neighbor. This information is stored in the Neighbor table. It is populated by the system manager, with one exception. During the join process and provisioning process, where the neighbor entry is created automatically from the advertisement, the EUI-64 shall be acquired by the DL through the acknowledgement as described in 9.1.10.1.
- GroupCode1, GroupCode2. Links with a matching group code may be used to address this neighbor. The scope of the group code is within a single device. When a link has a group NeighborType=2, the link designates a group code instead of a neighbor, and the link applies to any queued DPDU where the neighbor has a matching group code. This enables a single transmit link to be shared by a group of neighbors. A value of zero indicates that no group code applies. Support for group codes is mandatory in routers and optional in I/O devices. The presence or absence of this capability is reported to the system manager when the device joins the network through the field dlmo.DeviceCapability.Options (see 9.4.2.23).
- Clock source. If this indicator is >0, then the neighbor shall be a DL clock source for this device. A value of 1 indicates a secondary DL clock source, and a value of 2 indicates a preferred DL clock source. See 9.1.9.2.3.
- ExtGrCnt and ExtendGraph. See 9.1.6.3 for a discussion of graph extensions. See Table 170 for the fields in each graph extension entry in ExtendGraph. If the neighbors address matches the destination address encoded in the DADDR sub-header, and the Graph\_ID designated in ExtendGraph matches the DPDU's DL route, extend the designated graph to that neighbor for that DPDU. For each neighbor, up to three such graphs may be extended. Support for ExtendGraph is optional in a device, and the presence or absence of this capability is reported to the system manager when the device joins the network through the field dlmo.DeviceCapability.Options (see 9.4.2.23).

For each ExtendGraph entry, include:

- Graph ID is the 12-bit graph ID that is being extended. See 9.4.3.6.
- LastHop. If this indicator is 1, the DL shall only use links to the neighbor for applicable DPDU's. In this case, the DL shall treat the extended graph index as the single forwarding alternative.
- PreferredBranch. If this indicator is 1, the DL should treat this graph extension as the preferred branch for applicable DPDU's. (See PreferredBranch field in 9.4.3.6.2).

Graphs are extended implicitly whether ExtendGraph is designated or not. The optional explicit ExtendGraph feature is intended to support optimizations that seek to control the graph ID of this final hop, and/or designate it as the last hop or preferred branch.

- DiagLevel. If this indicator has any non-zero bits, then the DL shall collect link diagnostics for this neighbor in the read-only attribute dlmo.NeighborDiag. If Bit 0=1, summary diagnostics shall be accumulated. If Bit1=1, clock diagnostics shall be accumulated. See 9.4.3.9.
- LinkBacklog, LinkBacklogIndex, LinkBacklogDur, and LinkBacklogActivate provide an index to a link that may be activated through the DAUX sub-header (type 2). See 9.3.5.4 for a general description of link activation. The DL, when transmitting a DPDU to this neighbor, may detect a backlog of applicable DPDU's on its message queue, and therefore signal the neighbor to activate link Index=LinkBacklogIndex to receive DPDU's for the next LinkBacklogDur occurrences of the link (see 9.3.5.4). LinkBacklogActivate indicates the size of the applicable DPDU backlog on the queue, excluding the DPDU being transmitted, that should trigger sending the link activation DAUX, unless the link is already activated. The system manager, when it configures LinkBacklogIndex, LinkBacklogDur, and LinkBacklogActivate, should also configure a transmit link with the same index; so that, a receive link on the neighbor has the same index as a corresponding transmit link in the DL that originates the link activation message, with both being activated for the number of occurrences indicated by LinkBacklogDur. DPDU's queued to this neighbor should be given high priority for that link index during the period defined by LinkBacklogDur. When counting candidate DPDU's on the message queue, the DL should account for all queued DPDU's that can be addressed to the neighbor. The transmitter (the device that initiates activation of the idle links) should activate its own idle link for a count of LinkBacklogDur

transmission opportunities (link occurrences). The receiver (the device on which the receive idle link has been activated) should activate its idle link for LinkBacklogDur reception opportunities (link occurrences). Transmission and reception opportunities should be counted even if a higher priority link is actually used in a particular timeslot. Generally, the idle link should be configured with a high priority.

Table 170 specifies the fields for each element in the ExtendGraph sequence.

**Table 170 – ExtendGraph fields**

Field name	Field encoding
Graph_ID (*Index of dlmo.Graph attribute)	Type: Unsigned12
LastHop (indicates whether the neighbor shall be the last hop)	Type: Unsigned1 Valid value set: 0 to 1 0=no 1=yes
PreferredBranch (indicates whether to treat the neighbor as the preferred branch)	Type: Unsigned1 Valid value set: 0 to 1 0=no 1=yes
Reserved (octet alignment)	Type: Unsigned2=0

Table 171 specifies the ExtGraph structure.

**Table 171 – ExtGraph structure**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Graph_ID (bits 11-4)								
2	Graph_ID (bits 3-0)			LastHop	PreferredBranch	Reserved=0			

#### 9.4.3.4.3 dlmo.NeighborDiagReset

dlmo.NeighborDiagReset provides read/write access to the field of the dlmo.Neighbor attribute that is used to set the neighbor diagnostic level. It is conceptually similar to a SQL view of dlmo.Neighbor. It can be used to read and write a specific field within dlmo.Neighbor, but it shall not be used to add or delete entries.

Table 172 specifies the fields within a dlmo.NeighborDiagReset OctetString.

**Table 172 – dlmo.NeighborDiagReset fields**

Field name	Field encoding
* Index	Type: ExtDIUInt (used as an index) Valid value set: 1 to 32767
Reserved (octet alignment)	Unsigned4 = 0
DiagLevel (selection of neighbor diagnostics to collect)	Type: Unsigned3 Bit0 = 1 Collect summary information Bit1 = 1 Collect clock information
Reserved (octet alignment)	Type: Unsigned2 = 0

Table 173 illustrates the structure of dlmo.NeighborDiagReset.

**Table 173 – dlmo.NeighborDiagReset structure**

octets	bits								
	7	6	5	4	3	2	1	0	
1 - 2	* Index								
1	Reserved=0			DiagLevel			Reserved=0		
				Bit 1	Bit 0				

Fields are exactly as specified for the identically named fields in dlmo.Neighbor attribute, and the instantaneous values of these fields are the same as in dlmo.Neighbor.

#### 9.4.3.5 dlmo.Superframe

##### 9.4.3.5.1 General

dlmo.Superframe is an indexed OctetString collection that contains superframes. The superframe structure enables the system manager to connect a set of links (dlmo.Link) to a repeating schedule of timeslots (dlmo.Superframe) of fixed duration. The superframe also designates a baseline channel-hopping schedule for those links.

The system manager inserts, updates, or deletes dlmo.Superframe entries by sending the DMAP a superframe, along with a unique index and, if selected, a TAI cutover time.

Each superframe describes a schedule that is specified in reference to TAI time zero. Derivation of current timeslot state from the superframe definition is described in 9.4.3.5.3.

A superframe may be configured with randomized timings, intended exclusively for links that transmit or receive solicitations and/or advertisements. Since a randomized superframes schedule is not synchronized to its neighbors, such a superframe shall not be used to transmit payload in DSDUs.

##### 9.4.3.5.2 Semantics

Table 174 specifies the fields for dlmo.Superframe.

**Table 174 – dlmo.Superframe fields**

Field name	Field encoding
* Index	Type: ExtDUIuint (used as an index) Valid value set: 1 – 127
TsDur (duration of timeslots within superframe; timeslots are realigned with TAI time reference every 250 ms)	Type: Unsigned16 Units: 2 <sup>-20</sup> s
ChIndex (selects hopping pattern from dlmo.Ch)	Type: ExtDUIuint
ChBirth (absolute slot number where channel hopping pattern nominally started)	Type: Unsigned8
SfType (Type of superframe)	Type: Unsigned2 Valid value set: 0 to 3 0=Baseline 1=Hop on link only 2=Randomize slow hop duration 3=Randomize superframe period
Priority (priority to select among multiple available links)	Type: Unsigned4
ChMapOv (indicates whether to override ChMap default)	Type: Bit1 Valid value set: 0 to 1 1=ChMapOv transmitted and used 0=ChMapOv not transmitted and defaults to 0x7fff
IdleUsed	Type: Bit1 Valid value set: 0 or 1 1=IdleTimer transmitted and used 0=IdleTimer not transmitted and defaults to -1
SfPeriod (base number of timeslots in each superframe cycle)	Type: Unsigned16
SfBirth (absolute slot number where the first superframe cycle nominally started)	Type: Unsigned16
ChRate (indicates the number of timeslots per hop)	Type: ExtDUIuint Valid value set: 1 to n 1 = slotted hopping >1 = slow hopping
ChMap (channel map used to eliminate certain channels from the hopping pattern for spectrum management)	Type: Unsigned16 or null
IdleTimer (idle/wakeup timer for superframe)	Type: Integer32 or null See text
RndSlots (indicates extent of randomization, in number of slots)	Type: Unsigned8 or null

Table 175 illustrates the structure of dlmo.Superframe.

**Table 175 – dlmo.Superframe structure**

<b>octets</b>	<b>bits</b>							
	7	6	5	4	3	2	1	0
1 octet	* Index							
2 octet	TsDur (MSB)							
1 octet	ChIndex (range 1-127)							
1 octet	ChBirth							
1 octet	SfType	Priority					ChMapOv	IdleUsed
2 octet	SfPeriod (MSB)							
2 octet	SfBirth (MSB)							
1-2 octets	ChRate							
0 or 2 octets	ChMap (MSB)							
0 or 4 octets	IdleTimer (MSB)							
0-1 octet	RndSlots							

Fields include:

- Timeslot duration (dlmo.Superframe[].TsDur). All timeslots within a superframe have the same duration, and a device is not required to handle multiple timeslot durations at the same time. Timeslots shall be realigned to the TAI clock every 250 ms. See 9.1.9.1 for information on timeslot alignment.
- Channel hopping pattern identifier (dlmo.Superframe[].ChIndex). Select an available pattern from dlmo.Ch (see 9.4.3.2.2).
- Channel hopping birthday (dlmo.Superframe[].ChBirth). Specifies the starting point of the channel hopping pattern, as a timeslot offset from TAI=0. Calculation of current position in hop sequence is described in 9.4.3.5.3.
- Superframe type (dlmo.Superframe[].SfType) indicates the type of superframe. Handling of each superframe type is described in 9.4.3.5.3.
- Superframe priority (dlmo.Superframe[].Priority) indicates the priority of the superframe. A higher Priority value will give that superframes links a higher priority. See 9.1.8.5 (pseudocode) for additional information on priority levels.
- Channel map override default (dlmo.Superframe[].ChMapOv). Indicates whether to override ChMap default of 0x7fff. If ChMapOv=1, change the default based on ChMap.
- Superframe period (dlmo.Superframe[].SfPeriod) indicates the number of timeslots in each base cycle of the superframe. With 10 ms timeslots, a 16-bit superframe period supports superframes up to about 10 minutes long.
- Superframe birthday (dlmo.Superframe[].SfBirth) indicates the nominal starting point for the “first” superframe that began its first cycle soon after TAI=0. It provides the slot offset from absolute timeslot 0, which occurred at nominal time TAI=0. SfBirth shall equal ChBirth when SfType= 1. See 9.4.3.5.3.
- Channel hopping rate (dlmo.Superframe[].ChRate) indicates the number of timeslots per hop. A hopping rate of 1 indicates slotted hopping; a hopping rate of rate greater than 1 indicates some degree of slow hopping. ChRate shall =1 when SfType= 1.
- Channel map (dlmo.Superframe[].ChMap) is used to eliminate certain channels from the hopping pattern, thus shortening the hopping pattern. Bit positions 0...15 correspond to channels 0...15, where a 0 bit in any position indicates that the corresponding channel shall not be used by the superframe, with the channel hopping sequence shortened accordingly. This attribute shall not be transmitted and defaults to 0x7fff if ChMapOv=0 (i.e., optional channel 15 is excluded by default).
- Idle superframe timer (dlmo.Superframe[].IdleUsed, dlmo.Superframe[].IdleTimer) provides the system manager with control over when a superframe is activated or idle, where an idle superframe treats all of its links as idle. The system manager may set

IdleTimer through the dlmo.Superframe attribute or the dlmo.SuperframeIdle attribute. IdleTimer is not transmitted and defaults to a value of -1 if IdleUsed=0.

- When IdleTimer is set to a positive number, the superframe shall be active and IdleTimer shall be decremented by the DL each TAI second until it reaches a value of 0. When the value of IdleTimer is 0, the superframe shall be idle.
- When IdleTimer is set to a negative number that is less than -1, the superframe shall be idle and IdleTimer shall be incremented by the DL each TAI second until it reaches a value of -1. When the value of IdleTimer is -1, the superframe shall be active.
- Randomization of superframes is controlled by dlmo.Superframe[].RndSlots. RndSlots is meaningless and shall not be transmitted if dlmo.Superframe[].SfType<2. Randomized superframes are intended exclusively to enable randomized network discovery processes. For example, a device in the provisioned state may be configured with a randomized superframe that is used to search for advertisements from a target network. Such randomization can be used to guarantee that the scan's sleep cycle is not synchronized with the advertisement schedule of the target network, thereby ensuring that an advertisement is eventually received. Only receive links, dedicated advertisements, and solicitations should be configured for use with a superframe where RndSlots>0. All other links for such randomized superframes shall be treated as idle.
- When SfType=2, a randomized number of timeslots in the range of 0 to RndSlots shall be added to the end of each superframe cycle.
- When SfType=3, a randomized number of timeslots in the range of 0 to RndSlots shall be added to the length of each slow hopping period. If slotted hopping is used, each hop shall be treated as a slow hopping period extended by a randomized number of timeslots.

#### 9.4.3.5.3 Superframe current timeslot state

This subclause describes how the current superframe timeslot state shall be derived from the superframe fields. A result is described, not intending to constrain how a device internally arrives at these results. All features described in this subclause shall be supported by all DLs that comply with this standard, unless specifically designated as optional.

These derivations of the current timeslot state use fields in dlmo.Superframe[], which are based on the state at TAI time of zero or soon thereafter. Implementations may reasonably use these formulas to establish a starting state when the superframe is initialized, and then update that state incrementally going forward. However, the incremental update approach will not work when there is a change in any field used in the base calculation, or in fields in other attributes (dlmo.Ch[] in general, and dlmo.Link[] for SfType=1) that are used in the base calculation. When those fields are changed, the current state needs to be derived again.

**NOTE** The notion of an absolute timeslot is used here as a variable to calculate the current timeslot state. Other standards use an absolute timeslot to identify the timeslot; but here an absolute timeslot is used only as an intermediate value in a calculation, and is not referenced outside of this subclause.

Each TAI quarter-second period has a fixed number of timeslots that can be described by the formula:

$$\text{SlotsPerQuarterSec} = \text{Truncate}(2^{18} / \text{TsDur})$$

An absolute slot number (SlotNumAbs) can be derived from the scheduled start time of the current timeslot (ScheduledTaiTime), simplified by the fact that ScheduledTaiTime is required to be re-aligned to TAI time every quarter-second. The slot offset from TAI=0 can be derived accordingly:

$$\text{TaiQuarterSec} = \text{RoundDown}(\text{ScheduledTaiTime}, 0.25)$$

$$\text{SlotWithinQuarterSec} = (\text{ScheduledTaiTime} - \text{TaiQuarterSec}) / \text{TsDur}$$

$$\text{SlotNumAbs} = (\text{TaiQuarterSec} * 4 * \text{SlotsPerQuarterSec}) + \text{SlotWithinQuarterSec}$$

SfType=0 designates the baseline case, where all superframe cycles include a fixed number of timeslots and the channel hopping schedule also has a fixed cycle.

- The superframe provides a fixed superframe period (SfPeriod) which is the number of timeslots in each superframe cycle. It also provides an absolute slot number (SfBirth), following TAI=0, as a reference starting time for the first superframe. The superframe offset of the current timeslot is:

$$\text{SfOffset} = (\text{SlotNumAbs} - \text{SfBirth}) \bmod \text{SfPeriod}$$

- The channel hopping pattern nominally begins at absolute slot number ChBirth. The number of elements in the channel hopping schedule (ChCount) can be determined from the length of the channel hopping pattern selected by ChIndex, and subtracting the number of entries that are removed from the sequence as indicated by ChMap,
- The number of timeslots in a cycle of the hopping pattern depends on whether slow or slotted hopping is used, as indicated by ChRate.

$$\text{ChCycle} = \text{ChCount} * \text{ChRate}$$

- The timeslot offset into that channel hopping cycle is:

$$\text{ChOffset} = (\text{SlotNumAbs} - \text{ChBirth}) \bmod \text{ChCycle}$$

SfType=1 designates a variant of slotted hopping, where hopping occurs only when there is a link.

- Device support for SfType=1 is optional, as reported to the system manager through the attribute dlmo.DeviceCapability.Options (Bit 5). SfType=1 shall not be combined with slow hopping, i.e., ChRate=1.
- The superframe offset is as described in SfType=0.
- The number of hops per superframe cycle (ChPerSuperframe) is determined by counting the number of timeslots in each superframe cycle that are referenced by at least one link. This is not a simple count of links, because some links may refer to multiple timeslots, and some timeslots may be referenced by multiple links.
- Since the number of channel hops is a multiple of the number of superframes, the next step is to calculate the number of superframe cycles that have been completed since TAI=0.

$$\text{CurrentSfStartAbs} = (\text{SlotNumAbs} - \text{SfOffset})$$

$$\text{SfCyclesSinceBirth} = (\text{CurrentSfStartAbs} - \text{SfBirth}) / \text{SfPeriod}$$

- For SfType=1, the superframe cycle and channel hopping cycles are required to start at the same time (SfBirth=ChBirth). The channel offset at the start of the current superframe is determined by the formula:

$$\text{ChOffset}_{\text{StartSf}} = (\text{SfCyclesSinceBirth} * \text{ChPerSuperframe}) \bmod \text{ChLen}$$

- Starting from ChOffsetStartSf, the current hopping offset can be determined by stepping through the superframe from the start of the superframe cycle to the current timeslot. Hopping offset within each superframe cycle cannot be reduced to a simple linear formula since the links are not necessarily spread evenly through the cycle.

SfType=2 extends each slow hopping interval by a randomized number of timeslots in the range of 0 to dlmo.Superframe[].RndSlots. The initial starting point of the channel hopping sequence may also be randomized when SfType=2, and superframe timing shall be as described for SfType=0.

SfType=3 extends each superframe cycle by a randomized number of timeslots in the range of 0 to dlmo.Superframe[].RndSlots. The initial starting point of the superframe cycle should also be randomized when SfType=3, and the channel hopping sequence shall be as described for SfType=0.

#### 9.4.3.5.4 Slow hopping

Slow hopping is defined as a superframe where dlmo.Superframe[].ChRate>1, resulting in a set of contiguous links on the same channel. The channel hopping rate, ChRate, may be

configured as equal to the superframe period, SfPeriod, and in that case each hopping period may reasonably be configured as a range of links using dlmo.Link[].Schedule=2.

The receive side of a slow hopping configuration should use the default receive template for scanning per Table 167, or a similar template. This template, when applied to contiguous timeslots on the same channel, should run its receiver continuously, and may run a transaction to completion even if that transaction runs across the edge of a timeslot. A receive link using that template may repeat frequently or continuously within a superframe, typically with a low priority to give precedence to slotted hopping operations.

A set of slow hopping receive links on a given channel, using the default receive template for scanning, may be temporarily interrupted by higher-priority transactions, for example, as shown graphically in Figure 66. In the absence of such transactions, the receiver should run continuously across the timeslot boundaries in such configurations.

The transmit side of a slow hopping configuration should use a template appropriate for a transmit transaction on the subnet, such as the default transmit template per Table 166. The transmit link configuration should be configured to account for clock drift. For example, in a network with 10 ms timeslots, a particular device in a slow hopping configuration might be expected to experience clock drift of up to 15 ms between clock updates. In that example, the first and last two timeslots in each slow hopping period should not be designated as transmit links, thereby incorporating 20 ms guard times into the configuration.

The transmit side in a slow hopping configuration may designate specific timeslots for transmission, or alternatively it may designate a range of timeslots. When a range of timeslots is designated, the channel hopping rate should match the superframe period, and the transmit link should be designated as a range (dlmo.Link[].Schedule=2), shown graphically in Figure 72. In that configuration, the DL should treat each range as a single transmit opportunity, and select the transmit link within the range on a randomized basis.

#### 9.4.3.5.5 dlmo.Superframeldle

dlmo.Superframeldle provides read/write access to only the fields of the dlmo.Superframe attribute that relate to the idle superframe. It is conceptually similar to a SQL view of dlmo.Superframe. It can be used to read and write specific fields within superframe indexed OctetStrings, but cannot be used to add or delete entries.

Table 176 specifies the fields within a dlmo.Superframeldle OctetString.

**Table 176 –dlmo.Superframeldle fields**

Field name	Field encoding
* Index	Type: ExtDIUInt (used as an index)
Reserved (octet alignment)	Type: Unsigned7=0
IdleUsed (indicates whether the superframe is idle when the IdleTimer is zero)	Type: Unsigned1 Valid value set: 0 to 1 0=no 1=yes
IdleTimer (idle/wakeup timer for superframe)	Type: Integer32 or null

Table 177 illustrates the structure of dlmo.Superframeldle.

**Table 177 – dlmo.Superframeldle structure**

octets	bits							
	7	6	5	4	3	2	1	0
1	* Index							
1	Reserved=0							IdleUsed
0 or 4	IdleTimer							

Fields are exactly as specified for identically named fields in the dlmo.Superframe attribute, and the instantaneous values of these fields are identical to those in dlmo.Superframe.

#### 9.4.3.6 dlmo.Graph

#### **9.4.3.6.1 General**

`dlmo.Graph` is an indexed `OctetString` collection that contains graphs. On a particular device, a graph is simply a list of neighbors that can be used for the next hop when a graph is specified in the DROUT sub-header.

The system manager inserts, updates, or deletes dlmo.Graph entries by sending the DMAP a graph, along with a unique index and (if selected) a TAI cutover time.

Graph ID = 0 shall not be used, because this value is reserved as an indicator in the DROUT sub-header, as described in 9.3.3.6.

Graph IDs are limited to a range of 12-bit values, with a range of 1 to  $2^{12}$ . In source routing, these 12-bit graph IDs are encoded as 1010gggggggggg.

As described in 9.1.6.3, immediate neighbors are implicitly treated as covered by a graph, whether the neighbor is listed in the graph structure or not. When the DPDUs destination address is in a DL's neighbor table, the graph is automatically extended to cover that neighbor. Thus, even though the structure of `dimo.Graph` can support only a few neighbors, the graph can handle many more through such graph extensions.

**NOTE** Graph IDs should be unique within the scope of a DL subnet. However, duplicated graph IDs are not prohibited. When two graphs with the same graph ID intersect in a single device, they become a single graph.

#### 9.4.3.6.2 Semantics

Table 178 specifies the fields for dlmo.Graph.

**Table 178 – dlmo.Graph**

Field name	Field encoding
* Index	Type: ExtDUInt (used as an index) Valid value set: 1 to 4095
PreferredBranch (indicates whether to treat the first listed neighbor as the preferred branch)	Type: Unsigned1 Valid value set: 0 to 1 0=no 1=yes
NeighborCount	Type: Unsigned3 Valid value set: 0 to 4
Queue (allocates buffers in the message queue for DPDUs that are being forwarded using this graph)	Type: Unsigned4
MaxLifetime	Type: ExtDUInt
Neighbors (Index into dlmo.Neighbor; typically two or three neighbors in list for next-hop diversity)	Type: SEQUENCE OF ExtDUInt (SIZE(NeighborCount))

Table 179 illustrates the structure of `dlmo.Graph` in the case where each `ExtDLUInt` requires one octet.

**Table 179 – dImo.Graph structure**

Elements include:

- `dlmo.Graph[]`.PreferredBranch. If this indicator is 1, treat the first listed neighbor as the preferred branch, and the DL should wait until there is an opportunity to try at least one transmission along the preferred branch before attempting other alternatives. If this indicator is 0, do not give such preferential treatment to the first listed neighbor.
- `dlmo.Queue` allows the system manager to reserve up to 15 buffers of the message queue for DPDUs that are following the graph.
- `dlmo.Graph[]`.MaxLifetime (units  $\frac{1}{4}$  s). If this element is non-zero, the value of `dlmo.MaxLifetime` shall be overridden for all DPDUs being forwarded following this graph.
- List of neighbors (commonly two neighbors for next-hop link diversity).

#### 9.4.3.7 `dlmo.Link`

##### 9.4.3.7.1 General

`dlmo.Link` is an indexed OctetString collection that contains links. Each link refers to exactly one `dlmo.Superframe` entry.

The system manager inserts, updates, or deletes links by sending the DMAP a link, along with a unique index and (if selected) a TAI cutover time.

When a neighbor is referenced in a transmit link, DPDUs that refer to that neighbor are considered as candidates for the link. DPDUs that refer to the neighbor through the first entry in the DROUT sub-header, either directly by address or indirectly through a graph, shall be considered as candidates for the link. In addition, DPDUs that designate the neighbor as the destination address in the DADDR sub-header shall also be considered as candidates for the link. The exception is that certain links are designated exclusively for DPDUs following specific graphs, and only DPDUs with matching GraphIDs shall be considered as candidates for such links. If multiple DPDUs on the message queue are candidates for a given link, the DPDU is selected by priority as described in 9.1.8.5.

##### 9.4.3.7.2 Semantics

Table 180 specifies the fields for `dlmo.Link`.

**Table 180 – `dlmo.Link` fields**

Field name	Field encoding
* Index	Type: ExtDUInt (used as an index)
SuperframeIndex (reference to <code>dlmo.Superframe</code> entry)	Type: ExtDLUInt
Type (see Table 182 and associated text)	Type: Unsigned8
Template1 ( <code>dlmo.TsTemplate</code> reference to primary template)	Type: ExtDLUInt
Template2 ( <code>dlmo.TsTemplate</code> reference to secondary template)	Type: ExtDLUInt or null
NeighborType	Type: Unsigned2 Valid value set: 0 to 2
GraphType	Type: Unsigned2 Valid value set: 0 to 2
SchedType	Type: Unsigned2 Valid value set: 0 to 3
ChType	Type: Bit1 Valid value set: 0 to 1
PriorityType	Type: Bit1 Valid value set: 0 to 1
Neighbor (identify neighbor or group)	Type: ExtDLUInt or null
GraphID (12-bit identity of graph with exclusive or prioritized access to link)	Type: ExtDLUInt or null
Schedule (link schedule; see Table 184)	Type: See Table 184
ChOffset (select channel based on offset from superframes hop pattern)	Type: Unsigned8 or null
Priority (link priority)	Type: Unsigned8 or null Valid value set: 0000xxxx

Table 181 illustrates the structure of `dlmo.Link`. ExtDLUInt fields are shown as single octets.

**Table 181 – dlmo.Link structure**

octets	bits							
	7	6	5	4	3	2	1	0
1-2 octet	* Index							
1 octet	SuperframeIndex (range 1-127)							
1 octet	Type							
1 octet	Template1 (range 1-127)							
0-1 octet	Template2 (range 1-127)							
1 octet	NeighborType	GraphType	SchedType	ChType	PriorityType			
0-2 octet	Neighbor							
0-2 octet	GraphID							
1-4 octet	Schedule (see Table 184)							
0-1 octet	ChOffset							
0-1 octet	Priority							

Elements include:

- `dlmo.SuperframeIndex`. Indicates the superframe reference for the link.
- `dlmo.Link[].Type`. Indicates how the link is configured for transmission and/or reception, and/or neighbor discovery. See Table 182.
- `dlmo.Link[].Template1`. Primary timeslot template. See 9.4.3.3 for a discussion of templates.
- `dlmo.Link[].Template2`. Secondary timeslot template, for transmit/receive (TR) slots only, in combination with other link selections. Use Template2 as the receive template, if there is no DPDU in the queue for the primary template. Template 2 is transmitted and meaningful only for TRx links, that is, links where `Link[].Type` bits 6 and 7 both have a value of 1. See 9.1.8.5.
- `dlmo.Link[].NeighborType`, and `dlmo.Link[].Neighbor`. A neighbor is designated for transmit links. See 9.4.3.4 for a discussion of neighbors. When a neighbor is designated in a link, it may reference either a `dlmo.Neighbor` index or a group (`dlmo.Neighbor[].GroupCode`).
- If `dlmo.Link[].NeighborType=0`, `dlmo.Link[].Neighbor` is null, and not transmitted.
- If `dlmo.Link[].NeighborType=1`, `dlmo.Link[].Neighbor` designates index into the `dlmo.Neighbor` attribute.
- If `dlmo.Link[].NeighborType=2`, `dlmo.Link[].Neighbor` designates a group.
- `dlmo.Link[].GraphType`, `dlmo.Link[].GraphID`. PDUs following a particular graph may be given exclusive or priority access to certain transmit links. These fields, when so configured, limit link access to certain graphs, thereby connecting the link to a particular communication flow through the DL subnet. When GraphType is left blank, the transmit link is available to any PDU that is being routed through the link's designated neighbor. When GraphType is used, a particular graph is given exclusive or priority access to the link.
- If `GraphType=0`, the `GraphID` element is null and is not transmitted.
- If `GraphType=1`, the link is designated for exclusive use by a particular graph. Access to the link shall be limited to PDUs following that graph.
- If `GraphType=2`, PDUs with a matching graph ID are given priority access. PDUs following that graph should be given priority over other PDUs when the link is used.
- `dlmo.Link[].SchedType`, `dlmo.Link[].Schedule`. Indicates the timeslot position(s) of the link within each superframe cycle. The schedule may designate a fixed offset, a fixed set of intervals, a range, or a bitmap.
- 0=Offset only
- 1=Offset and interval
- 2=Range

- 3=Bitmap
- dlmo.Link[].ChType, dlmo.Link[].ChOffset. Indicates how the link's channel is selected.
- If dlmo.Link[].ChType=0, dlmo.Link[].ChOffset is null and not transmitted. Simply follow the superframe's baseline channel hopping pattern.
- If dlmo.Link[].ChType=1, add dlmo.Link[].ChOffset to the superframes current dlmo.Superframe[].ChOffset (modulo hopping pattern length, accounting for excluded channels), and select the channel accordingly.
- dlmo.Link[].PriorityType, dlmo.Link[].Priority. Indicates how the links priority is set. Link priorities are functionally described in 9.1.8.5.
- If dlmo.Link[].PriorityType=0, dlmo.Link[].Priority is null and not transmitted. For transmit links, use priority dlmo.LinkPriorityXmit. For receive links, use priority dlmo.LinkPriorityRcv.
- If dlmo.Link[].PriorityType=1, use the dlmo.Link[].Priority. If the link is both a transmit and receive link, use dlmo.Link[].Priority for transmissions, and dlmo.LinkPriorityRcv for reception.

Table 182 illustrates the structure of the field dlmo.Link[].Type.

**Table 182 – dlmo.Link[].Type structure**

octet s	bits							
	7	6	5	4	3	2	1	0
1	Transmit	Receive	Exponential backoff	Idle	Discovery 0=None 1=Advertisement 2=Burst advertisement 3=Solicitation	JoinResponse 1=Link used to send join response to neighbor	SelectiveAllowed 0=Not allowed 1=Allowed	

The link types are defined as follows:

- Bit 7: Transmit (T=1). Indicates transmission of payload.
- Bit 6: Receive (R=1).
- Bit 5: Exponential backoff (B=1). The B bit indicates whether the sender should apply exponential backoff rules for retries (shared), versus using the timeslot without regard to exponential backoff (not shared). See 9.1.8.2.
- Bit 4: Idle (I=1). When the idle bit is set to 1, the link shall be idle unless temporarily activated in conjunction with transmission of an activate DAUX subheader; see 9.3.5.4. When the idle bit is set to zero, link activation does not apply to the link. A timeslot designated as idle may include a neighbor reference. Even without a neighbor reference, it needs transmit and receive bits set as needed to refer to the timeslot template it will need when activated.
- Bits 3 and 2 (DD): Advertisement or solicitation configuration. If discovery is non-zero, the link should be used to transmit an advertisement or solicitation, even if there is no higher-order payload that needs to be sent. There are two types of advertisement links. Discovery=1 is a single advertisement transmitted with timing as defined in the timeslot template. Discovery=2 is a burst advertisement. When Discovery=2 and the link is a receive link (Bit 6 = 1), that is a signal to scan for a burst advertisement in the timeslot.
- Bit 1: JoinResponse (J=1). When a router proxies a join request for an immediate neighbor, it will eventually receive a message from the system manager to forward to the device that is joining. The router forwards these messages using links that are flagged with JoinResponse=1. A join response on the message queue can be identified by a DPDU header with a 64-bit destination address. A timeslot designated as supporting a join response may include a neighbor reference, thereby enabling the link to also be used for regular subnet traffic. Even without a neighbor reference, it needs the transmit bit (Bit 7) set.

- Bit 0: SelectiveAllowed (S=1). The DL may, without system manager direction, autonomously and selectively treat transmit links as idle if they occur on certain radio channels with a history of poor connectivity. This is a form of selective channel utilization, and is described in 9.1.7.2.4. The DL may skip links occurring on channels that it autonomously deems problematic due to a history of poor connectivity, potentially with granularity of a specific channel used to communication with a specific neighbor. In this manner, the device can save energy and reduce unnecessary interference with other users of the spectrum. However, the DL shall not skip links in this fashion when the link is flagged with SelectiveAllowed=0.

Table 183 shows allowed combinations of bits in dlmo.Link[].Type. Bits shown as X indicate that a value of 0 or 1 is allowed. For example, several combinations involving the transmit bit (7) show an X for the exponential backoff bit (5), indicating that such links might be configured as shared or not.

**Table 183 – Allowed dlmo.Link[].Type combinations**

Combination TRBIDDJS	Description
1X10001X	Join response
10XX000X	Transmit, no advertisement
11XX000X	Transmit/Receive, no advertisement
10X0010X	Transmit, advertisement
00000100	Dedicated advertisement
00001000	Burst advertisement
01001000	Burst advertisement scan
00001100	Solicitation
010X0000	Receive

A transmit/receive link combination is essentially a compressed representation of a transmit link and a separate receive link. If at least one outbound DPDUs on the queue matches the link, it shall be treated as a transmit link using the primary timeslot template. Otherwise, it shall be treated as a receive link using the secondary timeslot template, with priority dlmo.LinkPriorityRcv.

In a burst advertisement transmit link, the timeslot shall be filled with advertisements, with a time range as defined by the timeslot template. The first DPDUs shall begin within 96 µs (3 octets) of XmitEarliest, and repeat with a 192-384 µs (3-6 octets) of spacing between subsequent PPDUs, continuing until the beginning of the next DPDUs would exceed XmitLatest. Burst advertisements are described in more detail in 9.1.13.3.

For burst advertisement scan links, the DL should scan for burst advertisement that might or might not begin at any time during the timeslot.

Support for transmission of burst advertisements is required for routers, and support for reception of burst advertisements is required for all DLs. The DL allows for optimizations of a burst advertisement scan. A DL implementation might not take advantage of such optimization opportunities, thereby expending additional energy by operating its radio receiver for the full duration indicated by the timeslot template.

It is possible to configure channel-hopping patterns, superframe periods and link intervals so that the result is that only certain radio channels in the hopping sequence are actually used. For example, if a link repeats every 20<sup>th</sup> timeslot, and the channel hopping pattern includes 15 channels, then only 3 channels will actually be used by the link. To avoid such scenarios, the link interval and the channel-hopping pattern may be configured to be mutually prime with no common factors.

Table 184 specifies the different types of schedules for a given link. Links in a superframe are indexed based on the timeslot offset in each cycle, with the first timeslot in each cycle having an offset of zero.

**Table 184 – Values for dlmo.Link[].Schedule**

<b>Value for dlmo.Link[].Schedule</b>	<b>Element encoding</b>	<b>Description</b>
0=offset only	ExtDLUInt (offset)	Link occurs once at a fixed timeslot position in each superframe cycle.
1=offset and interval	ExtDLUInt, ExtDLUInt (offset, interval)	Link occurs multiple times in each cycle, first at the given offset and then repeating at an interval until the end of the cycle. Values are specified in number of timeslots.
2=range	ExtDLUInt, ExtDLUInt (first, last)	Link occurs at a range of slots in each superframe cycle, starting with the offset given by the first value and continuing until the offset given by the second value.
3=bitmap	Bit32	Bitmap covers first 32 timeslots in each superframe cycle. Link occurs in timeslots with a corresponding 1 bit set in the bitmap. Following LSB conventions, the bitmap is transmitted in DMAP messages with the bits 7-0 transmitted first and bits 31-24 transmitted last.

#### 9.4.3.8 dlmo.Route

##### 9.4.3.8.1 General

dlmo.Route is an indexed OctetString collection that contains routes. dlmo.Route describes available routes for DPDUs. When a DSDU comes down the protocol stack from the network layer, it receives a final destination address, along with a contract ID and a priority class. The DL maps the contract ID and destination address into a route, based on table lookups. The priority class from the network layer is simply copied to the DL header without being considered in route selection.

The system manager inserts, updates, or deletes routes by sending the DMAP a route, along with a unique index and (if selected) a TAI cutover time.

For a description of route selection, see 9.1.6.5.

##### 9.4.3.8.2 Semantics

Table 185 specifies the fields for dlmo.Route.

**Table 185 – dlmo.Route fields**

<b>Field name</b>	<b>Field encoding</b>
* Index	Type: ExtDIUInt (used as an index)
Size (size (number of entries) of Route attribute)	Type: Unsigned4 Valid value set: 1 to 15
Alternative	Type: Unsigned2 Valid value set: 0 to 3
Reserved (octet alignment)	Type: Bit2=0
ForwardLimit (Initialization value for the forwarding limit in DPDUs that use this route)	Type: Unsigned8
Route (series of routing destinations; if entry starts with 0, represents a unicast address; if entry starts with 1010, represents a graph)	Type: SEQUENCE OF Unsigned16 (SIZE (Size))
Selector (see text)	Type: Unsigned16 or null
SrcAddr (see text)	Type: ExtDIUInt or null

Table 186 illustrates the structure of dlmo.Route.

**Table 186 – dlmo.Route structure**

octets	bits												
	7	6	5	4	3	2	1	0					
1-2 octets	* Index												
1 octet	Size			Alternative			Reserved=0						
1 octet	ForwardLimit												
2 octets	Route <sub>0</sub> (MSB)												
...	...												
2 octets	Route <sub>Size-1</sub> (MSB)												
0 or 2 octets	Selector												
0 - 2 octets	SrcAddr												

The attribute `dlmo.Route[].Selector` depends on the setting of `dlmo.Route[].Alternative`:

- When `dlmo.Route[].Alternative=0`, select this route if `dlmo.Route[].Selector` matches ContractID and `dlmo.Route[].SrcAddr` matches the SrcAddr (source address) field in the DADDR sub-header. This alternative shall not be used unless the device is a backbone router. If `dlmo.Route[].Alternative>>0`, `dlmo.Route[].SrcAddr` is null and shall not be transmitted.
- When `dlmo.Route[].Alternative=1`, select this route if `dlmo.Route[].Selector` matches ContractID.
- When `dlmo.Route[].Alternative=2`, select this route if `dlmo.Route[].Selector` matches destination address.
- When `dlmo.Route[].Alternative=3`, use this route as the default. `dlmo.Route[].Selector` is null and shall not be transmitted.

`dlmo.Route[].Alternative` shall be applied in order, with lower numbered Alternatives given precedence over higher numbered alternatives. There should be no more than one `dlmo.Route` entry per ContractID/SrcAddr combination (`Alternative=0`), no more than one entry per ContractID (`Alternative=1`), no more than one entry per destination address (`Alternative=2`), and no more than one default (`Alternative=3`). If there are duplicates, the matching entry with the lowest index shall be selected.

#### 9.4.3.9 dlmo.NeighborDiag

##### 9.4.3.9.1 General

`dlmo.NeighborDiag` is an indexed OctetString collection that contains diagnostics for a set of neighbors. The attribute is read-only, with rows created as needed by the DL.

Each `NeighborDiag` entry comprises an array of one or two OctetStrings, with each entry corresponding to a different neighbor.

`NeighborDiag` entries are instantiated by the system manager, by setting `dlmo.Neighbor[].DiagLevel` bits to non-zero values. Iff (if and only if) Bit0=1, then summary diagnostics shall be collected for the neighbor, consolidated across all channels. Iff Bit1=1, then detailed clock diagnostics shall be collected for the neighbor, consolidated across all radio channels.

NOTE Individual channel diagnostics are collected through the attribute `dlmo.ChannelDiag`.

Diagnostics include counters and levels, that are accumulated as described in 9.1.15.3. Generally, counters are incremented by one in the course of successful or unsuccessful transactions, while RSSI (signal strength) and RSQI (signal quality) are levels that are accumulated as exponential moving averages.

`NeighborDiag` is reported in three general ways:

- Through the HRCO, the system manager can configure the device to report `NeighborDiag` periodically, such as every 30 minutes. Following each such report, on a per-entry basis,

NeighborDiag counts shall be reset to zero. Levels shall use the current value as a starting point for the next period.

- The system manager can read (poll) NeighborDiag as a read-only attribute on a per-entry basis. As in an HRCO report, counts shall be reset to zero when read.
- The DL can additionally be configured, through the dlmo.AlertPolicy attribute, to report NeighborDiag information when diagnostic values exceed a threshold. Only the row triggering the alert is reported. No values are reset.

Generally in this standard, an indexed OctetString's metadata capacity is reported as the number of rows. Since rows in NeighborDiag can have substantially variable lengths, metadata for NeighborDiag (DiagMeta) shall be reported in memory capacity in octets for the OctetStrings, with the convention that each ExtDlUInt field is assumed to consume two octets. A DL shall have the capacity for summary diagnostics (dlmo.NeighborDiag[].Summary) for at least half of its neighbor capacity as indicated by dlmo.NeighborMeta.Capacity, or for at least two neighbors, whichever is greater.

#### 9.4.3.9.2 Semantics

Each NeighborDiag entry includes three OctetStrings, one each for Summary and ClockDetail diagnostics. A zero-length OctetString indicates that the diagnostic is not being accumulated. Table 187 specifies the fields for dlmo.NeighborDiag.

**Table 187 – dlmo.NeighborDiag fields**

Field name	Field encoding
* Index	Type: ExtDlUInt (neighbor address, used as an index)
Summary	Type: OctetString
ClockDetail	Type: OctetString

Summary or ClockDetail diagnostics may be absent, as indicated by an OctetString of length zero.

Table 188 specifies the fields within the diagnostic Summary OctetString.

**Table 188 – Diagnostic Summary OctetString fields**

Field name	Field encoding
RSSI (level)	Type: Integer8
RSQI (level)	Type: Unsigned8
RxDPDU (count)	Type: ExtDlUInt
TxSuccessful (count)	Type: ExtDlUInt
TxFailed (count)	Type: ExtDlUInt
TxCNAACK (count)	Type: ExtDlUInt
TxNACK (count)	Type: ExtDlUInt
ClockSigma (level)	Type: Integer16

Table 189 specifies the structure of the diagnostic Summary OctetStrings.

**Table 189 – Diagnostic Summary OctetString structure**

octets	bits							
	7	6	5	4	3	2	1	0
1 octet	RSSI							
1 octet	RSQI							
1-2 octets	RxDPDU							
1-2 octets	TxSuccessful							
1-2 octets	TxFailed							
1-2 octets	TxCNA_Backoff							
1-2 octets	TxNACK							
2 octets	ClockSigma (MSB)							

Fields include:

- RSSI (signal strength): See 9.1.15.2 for discussion of RSSI units. RSSI is accumulated as an exponential moving average; see 9.1.15.3.
- RSQI (signal quality): See 9.3.5.5 and 9.1.15.2 for discussion of RSQI units. RSQI is accumulated as an exponential moving average; see 9.1.15.3.
- RxDPDU: Count of valid DPDUs received from neighbor, excluding ACKs, NACKs, and DPDUs without DSDU payloads.
- TxSuccessful: Count of successful unicast transmissions to the neighbor, where an ACK was received in response.
- TxFailed: Count of DPDU unicast transmissions, where no ACK or NACK was received in response.
- TxCNA\_Backoff: Count of unicast transmissions that were aborted due to CCA. These aborted transmissions are not included in TxFailed.
- TxNACK: Count of NACKs received, not included in TxFailed.
- ClockSigma: A rough estimate of standard deviation of clock corrections, in units of  $2^{-20}$  s. A value that roughly accounts for approximately 68% of clock corrections. ClockSigma is reset to zero whenever counters are reset.

If the DL autonomously treats a transmit link as idle, as described in 9.1.7.2.4, such skipped links shall not be counted in the neighbor diagnostics. However, such skipped links are reflected in channel diagnostics, as described in 9.4.2.27.

Table 190 specifies the fields within the diagnostic Clock OctetString.

**Table 190 – Diagnostic ClockDetail OctetString fields**

Field name	Field encoding
ClockBias (level, signed)	Type: Integer16
ClockCount (count)	Type: ExtDIUInt
ClockTimeout (count)	Type: ExtDIUInt
ClockOutliers (count)	Type: ExtDIUInt

Table 191 specifies the structure of the diagnostic ClockDetail OctetString, with ExtDIUInt fields shown as a single octet.

**Table 191 – Diagnostic ClockDetail OctetString structure**

octets	bits							
	7	6	5	4	3	2	1	0
2 octets	ClockBias (MSB)							
1-2 octets	ClockCount							
1-2 octets	ClockTimeout							
1-2 octets	ClockOutliers							

If the neighbor is a preferred DL clock source, as indicated by `IncludeClock=1`, it is recommended that the device be configured to accumulate the `ClockDetail` fields.

`ClockSigma`, `ClockBias`, and `ClockOutliers` all relate to clock corrections. If the neighbor is a DL clock source, these values relate to clock corrections received from the DL clock source. If the neighbor is not DL clock source, these values relate to clock corrections sent to the DL neighbor through the acknowledgement.

Fields include:

- `ClockBias`: An exponential moving average (EMA) of clock correction, in units of 2-20 s, including sign, with a 1% smoothing factor. See 9.1.15.3 for a discussion of EMA. If this value is significantly non-zero it indicates that the clock is biased.
- `ClockCount`: Count of clock updates received from or transmitted to the neighbor.
- `ClockTimeout`: Count of clock timeout events.
- `ClockOutliers`: Estimated count of clock corrections in excess of three standard deviations as per `ClockSigma`.

## 9.5 Data link layer methods

### 9.5.1 Method for synchronized cutover of DL attributes

A `Scheduled_Write` method, with `MethodID=1`, is provided to set an attribute at a specific TAI time. It exactly follows the template found in J.1.3.2, Table 517.

### 9.5.2 Methods to access indexed OctetString attributes

Various methods in the DL relate to writing, reading, and deleting indexed OctetString attributes. These methods are generally based on the templates provided in Annex J.

All indexed OctetString attributes in the DL are indexed by a single integer encoded as an `ExtDIUInt` (see 9.3.2.2). Following the convention of the template methods, these indexes are duplicated in the input arguments. For example, the `Write_Row` method includes an index as an input argument even though the index is also carried within the OctetString that constitutes the new entry.

Table 192 specifies the `Read_Row` method.

**Table 192 – Read\_Row method**

Method name	Method ID	Method description		
Read_Row	2	Method to read the value of a single row of an indexed OctetString attribute whose data is visualized as an information table		
<b>Input arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
1	Attribute_ID	Data type: Unsigned16	The attribute ID in the DLMO to which this method is being applied	
2	Index	Data type: Unsigned16	The * Index field in the attribute to access a particular row	
<b>Output arguments</b>				
Argument	Argument	Argument type	Argument description	

	#	name	(data type and length)	
	1	Data_Value	OctetString	An octet string that contains the contents of the row. If the row is empty, the OctetString shall contain only the Index, encoded as ExtDlUInt

Table 193 specifies the Write\_Row method.

**Table 193 – Write\_Row method**

Method name	Method ID	Method description		
Write_Row	3	Method to set / modify the value of a single row of an indexed OctetString attribute whose data is visualized as an information table		
<b>Input arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
1	Attribute_ID	Data type: Unsigned16 <given in management object definition>	The attribute ID in the DLMO to which this method is being applied	
2	Scheduled_TAI_Time	Data type: Unsigned32	TAI time in seconds at which the value should be written to the row of the structured attribute. If the time is in the past, the write shall be performed immediately	
3	Index	Data type: Unsigned16	The * Index field in the attribute to access a particular row	
4	Data_Value	OctetString	An octet string that contains the new contents of the row. If the DLMO row is unpopulated, a new row is created containing the OctetString if memory is available. If the DLMO row already exists, its contents are replaced with the OctetString. If the OctetString has a length of zero, then the row shall be deleted	
<b>Output arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
None				

Table 194 specifies the Write\_Row\_Now method. It is identical to the Write\_Row method, without the Scheduled\_TAI\_Time argument. It has the effect of writing an indexed OctetString row immediately on receipt.

**Table 194 – Write\_Row\_Now method**

Method name	Method ID	Method description		
Write_Row_Now	4	Method to set / modify the value of a single row of an indexed OctetString attribute whose data is visualized as an information table		
<b>Input arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
1	Attribute_ID	Data type: Unsigned16 <given in management object definition>	The attribute ID in the DLMO to which this method is being applied	
2	Index	Data type: Unsigned16	The * Index field in the attribute to access a particular row	
3	Data_Value	OctetString	An octet string that contains the new contents of the row. If the DLMO row is unpopulated, a new row is created containing the OctetString if memory is available. If the DLMO row already exists, its contents are replaced with the OctetString. If the OctetString has a length of zero, then the row shall be deleted	

			OctetString has a length of zero, then the row shall be deleted
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
None			

## 9.6 Data link layer alerts

### 9.6.1 DL\_Connectivity alert

DL performance diagnostics are accumulated in the attributes dlmo.NeighborDiag for per-neighbor diagnostics, and dlmo.ChannelDiag for per-channel diagnostics. Normally the system manager configures the HRCO to report these diagnostics periodically, and the DL automatically resets the diagnostic counters whenever these attributes are so reported. Between such reports, diagnostics may indicate a problem that needs to be reported to the system manager immediately. The DL\_Connectivity alert provides the mechanism for the DL to report such issues, and the dlmo.AlertPolicy attribute enables the system manager to set thresholds for such reporting.

The attribute dlmo.AlertPolicy enables/disables the DL\_Connectivity alert and provides thresholds to control whether alerts are reported. dlmo.AlertPolicy is an OctetString containing fields as shown in Table 195.

**Table 195 – dlmo.AlertPolicy fields**

Field name	Field scalar type
Descriptor (Enables or disabled the DL_Connectivity alert.)	Type Alert report descriptor (Table 257) Default: Disabled=True Default: Priority=0
NeiMinUnicast (Minimum number of unicast transactions needed for a neighbor report.)	Type: ExtDlUInt
NeiErrThresh (Report neighbor diagnostic if the percentage error rate reaches this threshold )	Type: Unsigned8
ChanMinUnicast (Minimum number of unicast transactions on a channel needed as a pre-condition for triggering an alert.)	Type: ExtDlUInt
NoAckThresh (Report ChannelDiag if a NoAck value is greater than this threshold)	Type: Unsigned8
CCABackoffThresh (Report ChannelDiag if a CCABackoff value is greater than this threshold)	Type: Unsigned8

Table 196 specifies the structure of the dlmo.AlertPolicy OctetString.

**Table 196 – dlmo.AlertPolicy OctetString structure**

octets	bits								
	7	6	5	4	3	2	1	0	
2 octets	Descriptor								
1-2 octets	NeiMinUnicast								
1 octet	NeiErrThresh								
1-2 octets	ChanMinUnicast								
1 octet	NoAckThresh								
1 octet	CCABackoffThresh								

Fields include:

- dlmo.AlertPolicy.Descriptor determines whether or not the DL\_Connectivity alert is enabled. By default, DL\_Connectivity alert is disabled until the system manager enables it by populating this attribute with appropriate thresholds. When Disabled=True, all other dlmo.AlertPolicy fields are meaningless and ignored.

- `dlmo.AlertPolicy.NeiMinUnicast` sets a minimum number of attempted unicast transactions before an error rate is considered significant. The count of attempted unicast transactions for a neighbor is the sum of the `dlmo.NeighborDiag` fields `TxSuccessful+TxFailed+TxCCA_Backoff+TxNACK`. If this sum is less than `NeighborTxMinReport`, do not send a `DL_Connectivity` alert for the neighbor.
  - `dlmo.AlertPolicy.NeiErrThresh` sets the threshold for reporting a `DL_Connectivity` alert for a neighbor. The percentage error rate is calculated as:

(TxFailed+TxCCA\_Backoff+TxNACK )\*100/( TxSuccessful+ TxFailed+TxCCA\_Backoff+TxNACK)

If this value is greater than NeiErrThresh, the diagnostics for the neighbor should be reported using the DL\_Connectivity alert unless there is an insufficient number of unicast transactions to the neighbor or the same alert has been recently reported.

- `dlmo.AlertPolicy.ChanMinUnicast` is similar to `NeiMinUnicast`. Counters underlying `dlmo.ChannelDiag` are not exposed, but a count of attempted unicast transactions is implicit in the reported ratios.
  - `dlmo.AlertPolicy.NoAckThresh` and `dlmo.AlertPolicy.CCABackoffThresh` provide thresholds for reporting. Since the values reported by `dlmo.ChannelDiag` are ratios, the reported values are simply compared to the thresholds. If the value exceeds the thresholds, `dlmo.ChannelDiag` should be reported through the `DL_Connectivity` alert unless the `ChanMinUnicast` requirement is not met or the same alert has been recently reported.

The system manager may respond to the DL connectivity alert by collecting diagnostics to more fully characterize the situation. Alternatively, particularly if a modified topology is easily achieved, the system manager may simply reconfigure the subnet topology.

Table 197 illustrates the structure of the DL\_Connectivity alert.

**Table 197 – DL\_Connectivity alert**

Standard object type name: dlmo (DL management object)					
Standard object type identifier: 124					
Defining organization: ISA					
Description of the alert: Poor neighbor connectivity					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority	Value data type	Description of value included with alert
Event	Comm	0 = DL_Connectivity	Medium	Type: OctetString	See Table 187
				Initial default value: N/A	
				Valid value set: Any valid 16-bit subnet address	

The format of the OctetString transmitted with the DL\_Connectivity Alert is shown in Table 198. It is simply the attribute number for either dlmo.ChannelDiag (48) or dlmo.NeighborDiag (46), followed by an OctetString containing the diagnostic data from that attribute. In the case of ChannelDiag, the entire attribute is transmitted. In the case of NeighborDiag, only the row that triggered the alert is transmitted, with the neighbor address specified within the row identifying the neighbor.

**Table 198 – DL\_Connectivity alert OctetString**

### 9.6.2 NeighborDiscovery alert

As described in 9.4.2.24, the NeighborDiscovery alert provides a mechanism for the DL to report the contents of the OctetString in dlmo.Candidates attribute.

Table 199 illustrates the structure of the NeighborDiscovery alert.

**Table 199 – NeighborDiscovery alert**

<b>Standard object type name: dlmo (DL management object)</b>					
<b>Standard object type identifier: 124</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Neighbor discovery alert</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority	Value data type	Description of value included with alert
Event	Comm	1 = NeighborDiscovery	Medium	Type: OctetString  Initial default value: N/A  Valid value set: N/A	An exact copy of the OctetString in dlmo.Candidates; see 9.4.2.24

## 10 Network layer

### 10.1 General

This clause provides an overview of the network layer functionality. It also describes services that the network layer offers to the layer above it (transport), the network layer management object (NLMO), and the structure of network protocol data units (NPDUs) and the formats of the various types of headers used.

The network layer header formats in this standard have been influenced by the IETF work group 6LoWPAN and an attempt is made to facilitate future compatibility.

The network layer follows the big endian convention; multi-octet fields are documented and transmitted with the high-order octet first (since they are treated as a series of octets by the lower layer). Within an octet, bits are documented starting from the high-order bit (bit 7) on the left and continuing to the low-order bit (bit 0) on the right.

NOTE The PhL (IEEE Std 802.15.4 radio) will transmit bits within an octet in little endian order, i.e., with the low-order bits first.

### 10.2 Network layer functionality overview

#### 10.2.1 General

The network layer in this standard performs the following functions:

- Addressing: The network layer determines the appropriate address information for a protocol data unit (PDU).
- Address translation: This standard uses primarily two types of addresses, short (16-bit) DL addresses, and long (128-bit) addresses. The short addresses are used within a DL subnet to conserve energy and bandwidth. Application endpoints and backbone networks use long addresses. The network layer is responsible for translation between the various types of addresses, e.g., when a PDU moves from a DL subnet to a backbone network (or vice versa).
- PDU formats: This standard allows for more than one NPU format to accommodate conservation of energy and bandwidth (which favors short headers), a variety of network topologies and internetworking with backbone networks. The network layer selects an appropriate format for the PDU based on such considerations as addressing, routing, level of service, etc.
- Fragmentation and reassembly: This standard handles fragmentation and reassembly at the network layer. An NPU of length more than the maximum data link service data unit (DSDU) size shall be fragmented by the network layer of the point of ingress into a DL subnet. Reassembly shall be performed by the network layer of the point of egress from the DL subnet.
- Routing: This standard performs routing at two levels: the backbone level and the mesh level. The network layer is responsible for routing PDUs at the backbone level. Routing at the mesh level is performed by the DL.

#### 10.2.2 Addressing

Applications in this standard use 128-bit addresses. Each device shall have a 128-bit address. If the device does not have a 128-bit address prior to the join process, the device shall be assigned such an address by the system manager during the joining process. The network layer uses these 128-bit addresses, but does not associate any further meaning with them.

Each device compliant with this standard shall also have a 128-bit address that is autoconfigured by the device as part of the initialization of its networking protocol stack. This 128-bit address is referred to as the device's link local address and shall be derived from the EUI-64. The format of this 128-bit address shall be that of a link local unicast address, as defined in IETF RFC 4291, 2.5.6. Table 200 illustrates this address structure.

**Table 200 – Link local address structure**

<b>10 bits</b>	<b>54 bits</b>	<b>64 bits</b>
1111111010	0	EUI-64

When PDUs are transmitted over a DL subnet, 128-bit addresses can consume valuable energy and bandwidth. Thus, this standard defines 16-bit aliases for 128-bit addresses; the short aliases are used over the DL subnet. For each DL subnet, a unique 16-bit address shall be assigned to every device within that subnet, as well as to every device outside the subnet with which a device within the subnet has a contract. This allows short addresses to be used in the DL subnet for all origins and destinations.

The scope of a 16-bit DL address is the DL subnet within which it has been defined. Thus, a particular device may have one DL address in the DL subnet to which it belongs and a different DL address in a foreign subnet. When a 16-bit DL address is used, it shall be carried in the data link protocol data unit (DPDU) header.

During the joining process, a device might not yet have a 128-bit address or a 16-bit DL address. In this case, messages between the joining device and the advertising router shall use the link-local addresses when needed (e.g., for the TSS pseudo header in the TL). The joining device and the advertising router shall be identified using their EUI-64s in the DPDU header.

The system manager assigns the 16-bit DL address and the 128-bit address of the devices operating in an ISA100.11a network. These addresses are assigned during the join process.

Backbone and plant network technologies are outside the scope of this standard; the standard does not specify the representation of the network layer 128-bit addresses on a particular backbone or plant network.

The network layer in this standard supports unicast addressing only. The 16-bit DL unicast addresses shall be in the range 0x0001-0x7FFF. Address 0x0000 shall be reserved to indicate no address. Addresses in the range 0x0001-0x007F can be further compressed by the DL (see 9.3.2.2).

NOTE Although multicast addressing is not supported in this standard, future releases may support multicast addressing.

### 10.2.3 Address translation

Since this standard employs 16-bit DL addresses within a DL subnet, when a PDU moves from a DL subnet to a backbone network (or vice versa), the network layer of the backbone router shall translate between 16-bit DL addresses and 128-bit network addresses. The same kind of translation shall be performed by the network layer of a DL subnet endpoint.

All devices in this standard shall maintain an address translation table (ATT), as shown in Table 201.

**Table 201 – Address translation table (ATT)**

<b>Short address (16-bit DL address)</b>	<b>Long address (128-bit network address)</b>
N1_16	N1_128
N2_16	N2_128
GW_16	GW_128
BBR_16	BBR_128
SM_16	SM_128

The address translation table is initialized during the join process with the DL address and the 128-bit address of the system manager. This information is part of the non-security join response received from the system manager as described in 6.3.9.2.

The address translation table shall be updated by the source device whenever a communication session is established with a new destination device. Communication sessions are described in 6.3.11.2.5.2. The DL address and the 128-bit address of the destination device will be stored in the address translation table upon the successful completion of the session establishment process. The process of session establishment is described in 7.5. If a session is terminated for whatever reason the entry associated with the destination device shall be deleted.

A device maintains entries in its ATT for other devices with which it communicates; these other devices may either belong to the same DL subnet as the first device or have a 16-bit DL address in the same DL subnet.

Within a particular DL subnet, a device (whether local or remote) shall have only one 16-bit DL address. Thus, the ATT can be used for both forward and reverse lookup by the network layer:

- 128\_bit\_address = ATT\_lookup (16\_bit\_DL\_address)
- 16\_bit\_DL\_address = ATT\_lookup (128\_bit\_address)

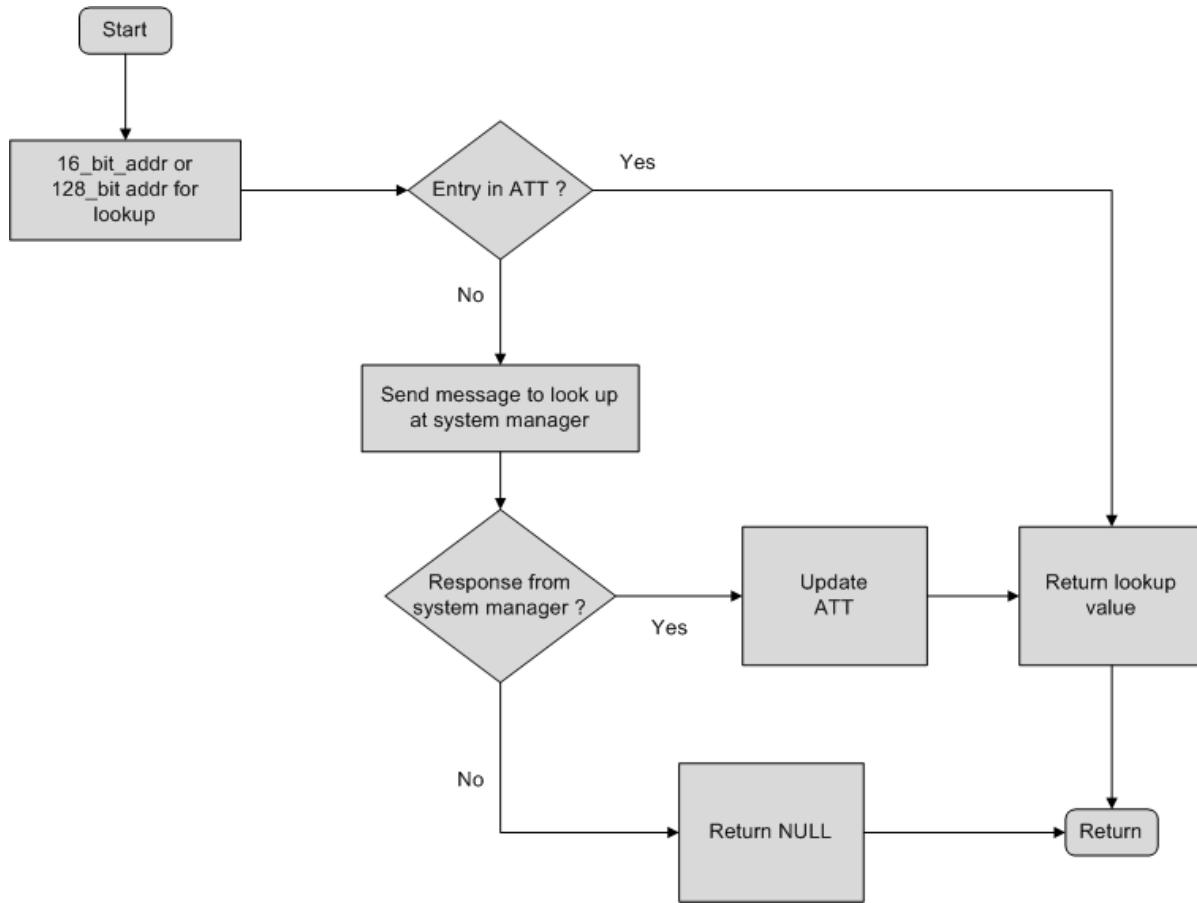
NOTE Multiple instances of the communication protocol suite of this standard can be co-packaged in a single physical device to support multi-homing. Although such operation is not specified by the standard, it is not prohibited.

An address with no entry in the ATT shall be translated with the help of the system manager. For each device joining the network, the system manager shall maintain the 128-bit network address of the device and its 16-bit DL address for each DL subnet in which the device has such an alias. Hence, the local ATT at a device shall be updated through the system manager.

The ATT is an integral part of the NLMO and can be directly updated by the system manager by utilizing the NLMO manipulation methods described in Table 210.

If a lookup in the ATT yields no results, then the lookup function shall notify the NLMO. The NLMO shall issue a read primitive to the directory service object (DSO) in the system manager to obtain the appropriate translation. The lookup function shall return a value of null if the system manager also has no mapping for a particular address or the system manager is not available. Any new information from the system manager shall be stored in the ATT table.

This process is illustrated in Figure 91.

**Figure 91 – Address translation process**

#### 10.2.4 Network protocol data unit headers

Three formats are used for NPDU headers with the first octet of the header being a dispatch octet used to distinguish between these formats:

- The basic header is intended for PDUs traversing a single DL subnet and shall be used only over the DL subnet. It is expected to be the most common format and minimizes the overhead associated with the transmission of headers. The basic header is just an abbreviation for a certain value of the 6LoWPAN compressed header; this value indicates that the source and destination addresses are elided, since they are carried by the DPDU header (see 10.5.2 for detailed information).
- The contract-enabled header shall also be used only over a DL subnet. It enables the originating device to include more information, e.g., the contractID. This information allows backbone routers to select appropriate resources (e.g., graphID, priority) for the routing of the PDU (see 10.5.3 for detailed information).
- The full header is the IPv6 header and is intended for use over the backbone. PDUs containing the basic or the contract-enabled header shall be expanded into the full header for routing over the backbone. Backbone routers convert full headers into basic headers for transmission over a DL subnet (see 10.5.4 for detailed information).

#### 10.2.5 Fragmentation and reassembly

If the entire NPDU is smaller than the maximum DSDU size, the NPDU shall not be fragmented and the network header shall not contain a fragmentation header. If the NPDU exceeds the maximum DSDU size, the NPDU shall be broken into fragments equal to or smaller than the maximum DSDU size. Fragmentation shall be performed by the network layer of the point of ingress into a DL subnet. Reassembly shall be performed by the network layer of the point of egress from the DL subnet.

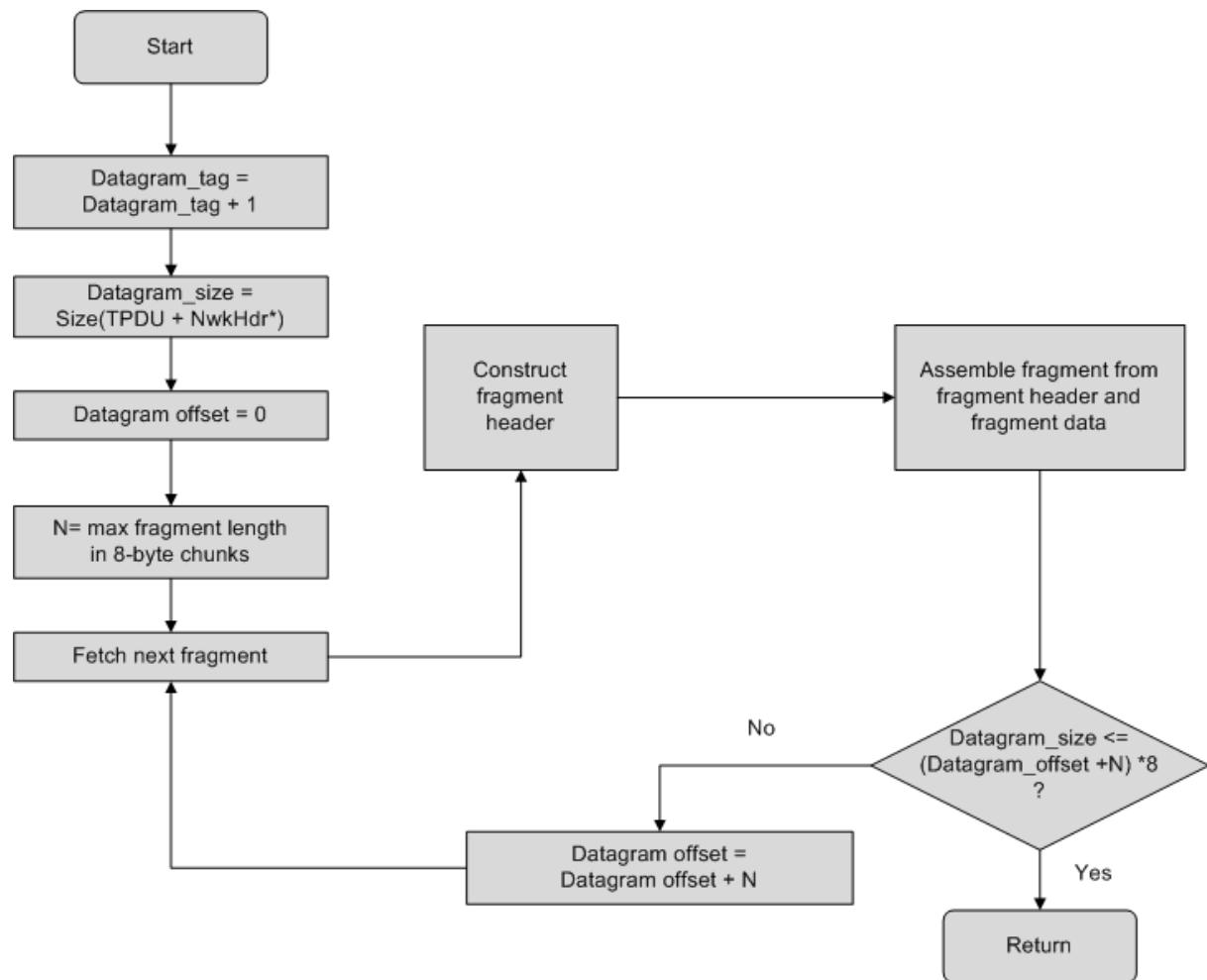
The first fragment shall contain the first fragment header as defined in Table 219. The second and subsequent fragments (up to and including the last fragment) shall contain a

fragmentation header as defined in Table 220. The offset of this fragment, referred to as the datagram offset, shall be expressed in units of eight octets.

The Datagram\_size field shall be present in every fragment to ease the task of reassembly in the event that a second (or subsequent) fragment arrives before the first. The inclusion of the Datagram\_size in every fragment allows the receiver to allocate the appropriate amount of buffer space even if the first fragment is delayed.

All fragments (first and subsequent fragments) shall have a Datagram\_tag field in their header. The value of this field shall be assigned by the device performing the fragmentation and shall be the same for all fragments of an NPDU, so that the reassembling device can recognize that the fragments belong to the same NPDU. The device performing the fragmentation shall assign a different Datagram\_tag value to fragments of different NPDU's. To achieve this, each device shall have a counter that is initialized to a random value and is incremented for every NPDU that undergoes fragmentation; the value of this counter shall be placed in the Datagram\_tag fields of all fragments of an NPDU.

Figure 92 illustrates the fragmentation process.



\*NwkHdr excluding fragmentation header

**Figure 92 – Fragmentation process**

To identify all fragments that belong to the same NPDU, the reassembling device shall use:

- The source address;
- Datagram\_tag;
- Datagram\_size; and

- The destination address.<sup>4</sup>

Upon receipt of a fragment, the network layer begins reconstructing the original unfragmented NPDU, whose size is Datagram\_size. The recipient shall use the Datagram\_offset field to determine the relative location of the individual fragments within the original unfragmented NPDU.

When a device first receives a fragment with a given Datagram\_tag, it starts a reassembly timer. If this timer expires before the entire NPDU has been reassembled, the received fragments shall be discarded. The reassembly timeout shall be set to a value defined in nlmo.Frag\_Reassembly\_Timeout (attribute identifier 11 in Table 206). If a fragment that partially overlaps another fragment is received, and it differs in either the size or Datagram\_offset of the overlapped fragment, the fragment(s) already accumulated in the reassembly buffer shall be discarded.

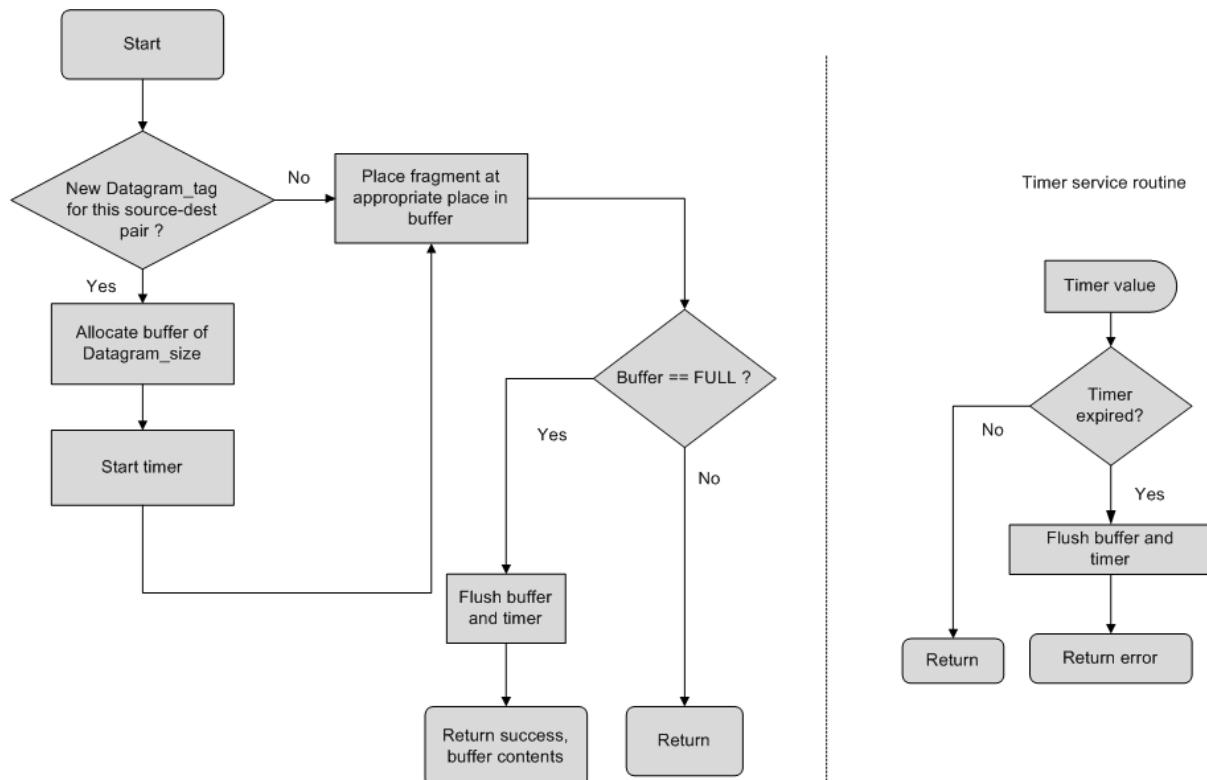
A new reassembly commences with a fragment containing a tag for which no fragments are pending. This may lead to buffers being allocated when some fragments arrive after the timeout of the reassembly process that had been previously initiated for the same tag. These new buffers will eventually be flushed after another nlmo.Frag\_Reassembly\_Timeout.

The reassembly process is completed when the NPDU is fully reassembled or the timer expires. If the NPDU exceeds the size indicated by nlmo.Max\_NSDU\_size, the reassembly process may be aborted and the NPDU may be discarded. The device may send a dropped PDU/PDU error alert with value 7 indicating that it is out of memory. Dropped PDU/PDU error alerts are shown in Table 211.

The NPDU reassembly process is shown in Figure 93.

---

<sup>4</sup> In the extremely rare case that two NPDU's from the same source to the same destination are fragmented by different intermediate routers that coincidentally pick the exact same Datagram\_tag the reassembling device may not be able to disambiguate fragments. In this case, the GraphID may be used to disambiguate further; however, this is not specified as mandatory in this standard. The TPDUs will be dropped due to checksum errors and retransmitted if this case does occur. Intermediate routers that fragment NPDU's may also coordinate their fragmentation state machines in order to avoid scenarios in which the reassembling device might not be able to disambiguate fragments.

**Figure 93 – Reassembly process**

## 10.2.6 Routing

### 10.2.6.1 General

Routing within a network compliant with this standard happens at two levels:

- One level comprises the endpoints and the backbone devices, if any; the network layer is responsible for routing PDUs at this level. This level does not handle routing over the DL links; traversal of a DL subnet appears as a single hop to this level.
- The second level of routing is within a DL subnet. This level is the responsibility of the DL layer (a layer 2 mesh implementation).

The routing between DL subnets and backbone networks is the responsibility of the NL. The backbone and plant networks are outside the scope of this standard; thus, this standard does not specify the details of how to route traffic over a particular backbone or plant network technology. However, this standard does specify the interfaces for data exchange between the backbone and the network layer of this standard. This standard also specifies minimum requirements for routing, along with management services for adding, deleting, and maintaining routes.

### 10.2.6.2 Routing tables

The network layer in devices compliant with this standard shall maintain a routing table (RT) to keep track of the next hop for a given destination. This table shall be maintained using 128-bit network addresses, since such addresses are unique across an entire network compliant with this standard (including all DL subnets). An example of a routing table is provided in Table 202. The routing table may be updated at the source device whenever a communication session is established with a destination device.

**Table 202 – Example of a routing table**

DestinationAddress	NextHop	NWK_Hop_Limit	OutgoingInterface*
N1	BBR1	2	Backbone
N2	BBR1	2	Backbone
GW	GW	2	Backbone
N3	N3	1	DL
N4	N4	1	DL
N5	N5	1	DL
...	...	...	...

\* This field will be set to DL for all destinations in routers and I/O devices.

**NOTE** In this standard, the route table and all NL management objects are specified to support only one active DL subnet at a time. All 16-bit DL addresses are unique within the scope of that single subnet. This is not intended to prevent a device from participating in multiple subnets simultaneously. Multiple subnets might be modeled as multiple instances of the NL, but such operation is not specified by this standard.

Field devices that are not backbone capable only route packets within the DL subnet. Routing within the DL subnet is the responsibility of the DL layer (a layer 2 mesh implementation). Hence field devices that operate in the DL subnet but are not backbone capable may maintain a routing table but are not required to do so. This is also reflected in Table 495 present in Annex B that normatively presents the minimum routing table sizes that need to be supported by devices that meet various role profiles. Network layer routing tables provide layer independence and allow potential route over implementations, where routing within the DL subnet is achieved through network layer routing.

The routing table shall also be used by the backbone routers to decide whether to route a PDU over the backbone or over the DL subnet of this standard. The OutgoingInterface field indicates whether the PDU shall be sent over the backbone or over the DL subnet.

NextHop indicates the next device whose network layer shall process the PDU destined for the DestinationAddress. Any device reachable through the DL mesh has NextHop equal to the destination address and the NWK\_Hop\_Limit field set to 1. From the perspective of the network layer, any device that is reachable through the DL mesh is a single network hop away.

#### **10.2.6.3 Processing of a network service data unit received from the transport layer**

When an NSDU is passed to the network layer from the transport layer, the NL determines the final destination for that NSDU based on the ContractID. The contract table (see Table 207) is used to obtain the destination address. Devices with both a backbone and a DL interface compliant with this standard shall look up the destination address in the routing table to determine which network interface to use. Field devices and field routers shall always use their DL interface.

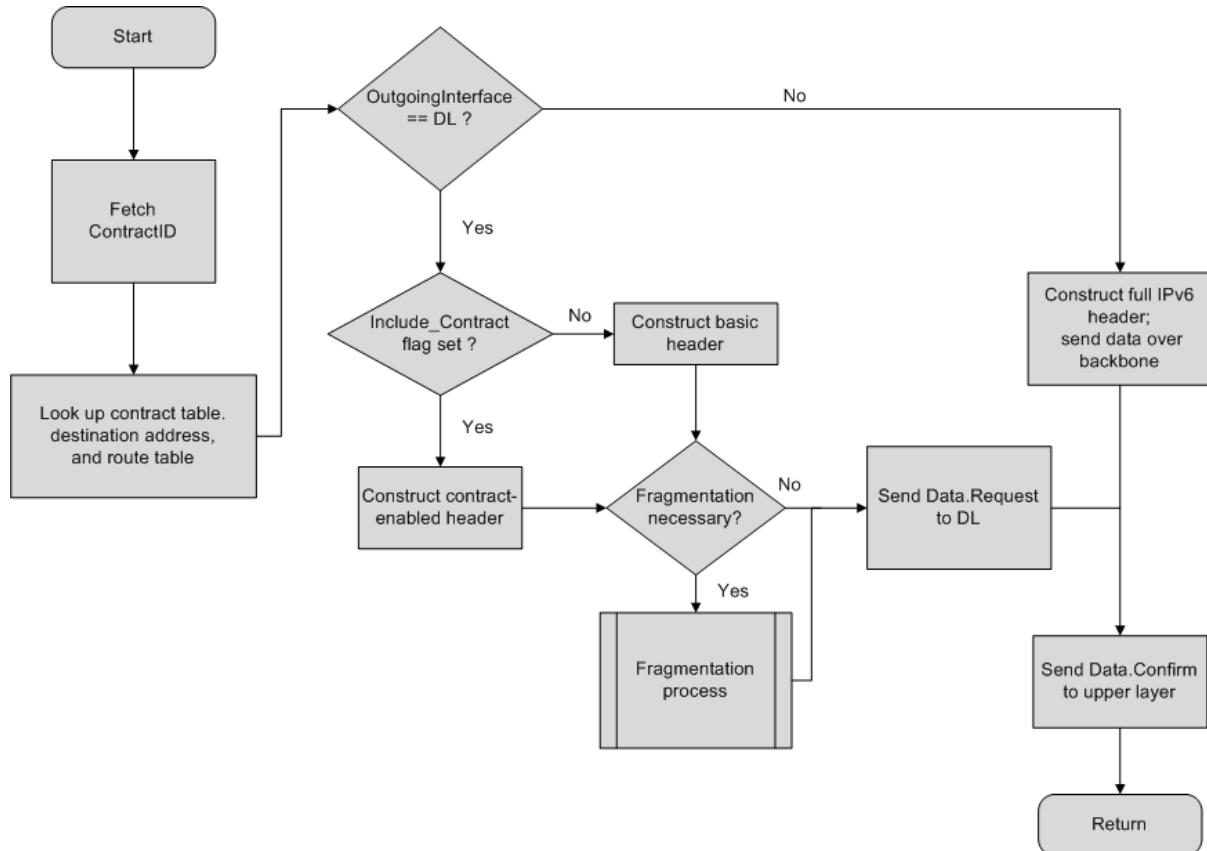
The network layer shall use the ContractTable to obtain the two-bit priority for the contractID in the N-Data.request. This contract priority shall be combined with the two bits of message priority (also passed in the N-Data.request) to obtain a 4-bit NPDU priority that is passed down to the DL; the two most significant bits shall be the contract priority, and the two least significant bits shall be the message priority. The Discard Eligible (DE) field from the N-Data.request is also passed down to the DL. If the OutgoingInterface for the destination address is the Backbone then the 4-bit priority and DE eligible bits shall be included in the TrafficClass field of the IPv6 header.

The network layer shall use the ContractTable to check if the ContractID needs to be included in the NPDU. Including the ContractID in the NPDU allows intermediate backbone routers to make appropriate routing choices (level of service, graphID, etc.) on the backbone or a different DL subnet. When routing over the DL interface, if ContractID need not be included, then a basic NPDU header should be constructed; otherwise, a contract-enabled NPDU header should be constructed.

The network layer shall also determine whether fragmentation is needed for the NPDU and shall perform the fragmentation process if necessary. Fragmentation shall be required only for NPUDUs routed over a DL subnet; dlmo.MaxDSDUSize shall indicate the maximum payload

that can be carried over the DL subnet. If the DSDU size is greater than this value, then fragmentation is necessary.

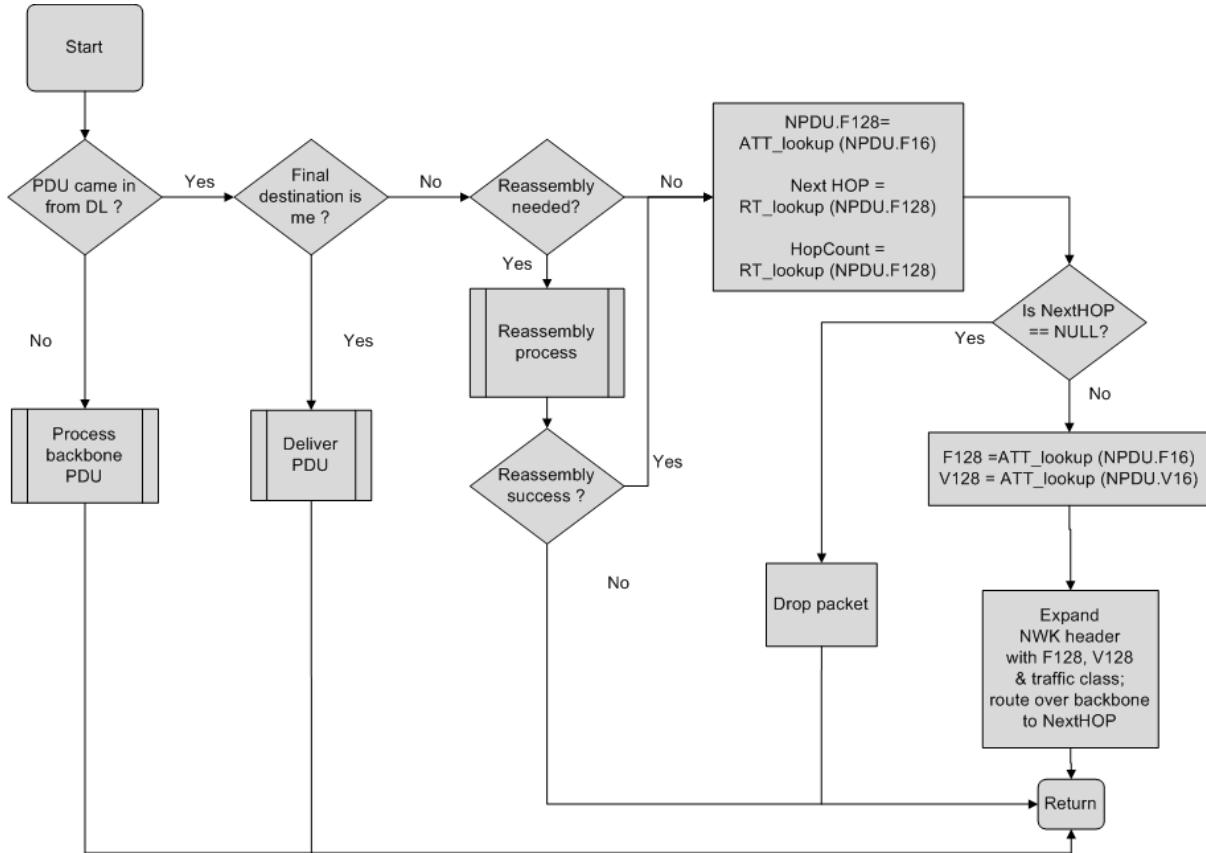
Figure 94 illustrates the processing of a NSDU received from the transport layer.



**Figure 94 – Processing of a NSDU received from the transport layer**

#### 10.2.6.4 Processing of an incoming network protocol data unit

An incoming NPDU (whether received from the DL or the backbone interface) shall first be checked to determine if the final destination is the current device. If so, the NSDU that is the payload of the NPDU shall be passed up to the transport layer. If the final destination is not the current device, then the device shall route the NPDU appropriately (via either the backbone or the DL interface). The overall decision process is shown in Figure 95. Not all packets received from the DL will have a corresponding 16 bit DL address entry in the ATT. Some devices operating on the backbone may not have an assigned 16-bit DL address, but only a 128-bit address. In such case the packet will be received from the DL forwarded over a default route. In that case the backbone capable device will directly look up the route associated with the destination 128-bit address.



**Figure 95 – Processing of an incoming NPDU**

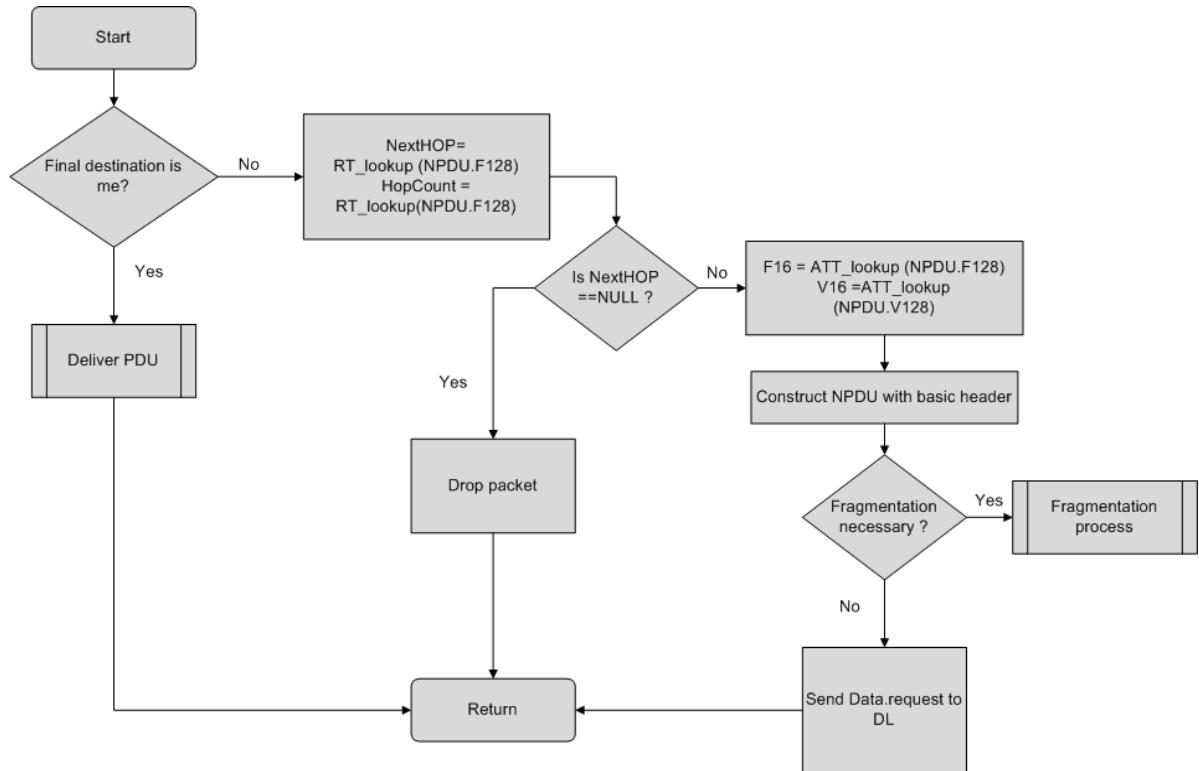
The DL layer's DD-DATA.indication and DD-DATA.request services provide a LastHop (LH) field. When this single-bit LH field is set to 1, it shall indicate that the PDU entered the DL subnet through a backbone router, and therefore shall not exit the DL through a backbone router to avoid circular routes at the network layer. This enables the network layer to elide the Hop Limit field and avoid circular routes when the basic header is used.

When the NPDU is received from the DL at a device other than the destination, if the last hop (LH) bit is set to 1 in DD-DATA.indication, this implies that the NPDU has reached the current device in error and the NPDU shall be dropped. If the NPDU is received from the DL subnet (see Figure 95), the intermediate router shall first check to see whether reassembly is needed for this NPDU. Once the reassembly is complete, the NPDU shall be prepared for routing over the backbone. The 16-bit DL address of the origin (very first V) and destination (final destination F) in DD-DATA.indication shall be translated into the 128-bit network addresses. Then the routing table shall be used to determine the next network layer hop for reaching the final destination. The network header shall be fully expanded and the NPDU shall be presented to the backbone interface for routing. As part of this expansion, the explicit congestion notification (ECN) value provided by the DD-DATA.indication shall be included in the appropriate field of the expanded header. This standard does not specify how the backbone handles and routes the NPDU. The backbone has the responsibility to deliver the NPDU to the NL of the NextHop.

This standard always uses ECN. When the notification is carried in the DL header, if the ECN bits are present in the network header, they shall be set to zero to indicate that the notification is carried in the DL header; a backbone router that receives a PDU from the DL shall use the ECN information carried by the DL header to fill in the ECN bits in the expanded header. PDUs originating from backbone devices shall have the ECN bits set to indicate that explicit congestion notification is used.

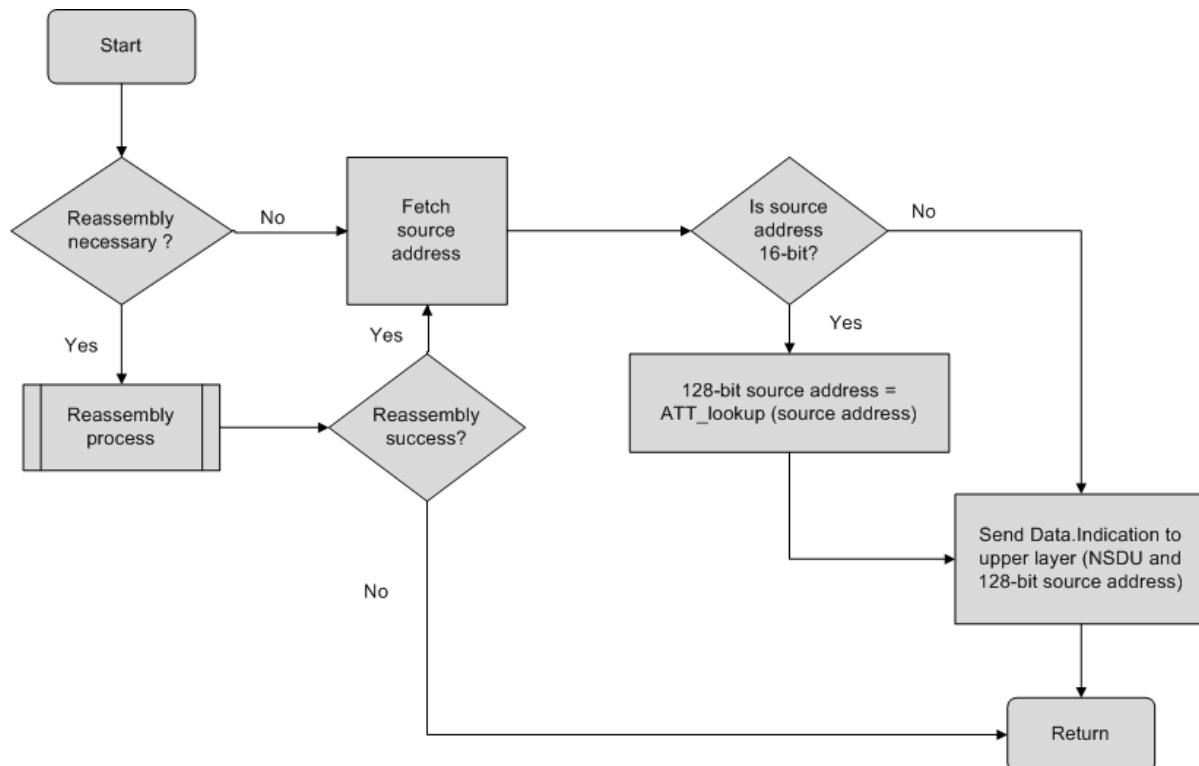
If the NPDU is received from the backbone, then it shall have an expanded header, and the final destination and very first (originator) addresses shall already be expressed as 128-bit network addresses. If the NPDU needs to be routed over a DL subnet, the 16-bit DL address in that DL subnet of the very first (originator) device and final destination shall be obtained

from the ATT and passed to the DL in DD-DATA.request. The network layer shall check if the ContractID and priority are included in the FlowLabel and TrafficClass fields, respectively, of the expanded header. If so, the ContractID and priority shall also be passed to the DL to allow the selection of appropriate routing mechanisms (GraphID, etc.). The presence or absence of congestion will be determined from the NPDU received from the backbone and will be passed to the DL in DD-DATA.request. When passing the NPDU with the basic header to the DL, the last hop (LH) parameter shall be set to 1 in DD-DATA.request; this indicates that the NPDU has entered a DL subnet and shall not be allowed to exit. If the NPDU size exceeds dlmo.MaxDsduSize for this DL subnet, the NPDU shall be fragmented. This process is depicted as a flowchart in Figure 96.



**Figure 96 – Processing of a NPDU received from the backbone**

If the receiving device is the final destination, then the network layer shall process the NPDU and shall pass the NSDU up to the transport layer, along with an indication of whether congestion was encountered (ECN bits). The network layer shall first check if it has received a fragment; if so, it shall perform the reassembly process (see Figure 93). The source address shall be translated into the 128-bit address, if necessary, and the NSDU shall be passed to the transport layer. Figure 97 depicts the flowchart for this processing.

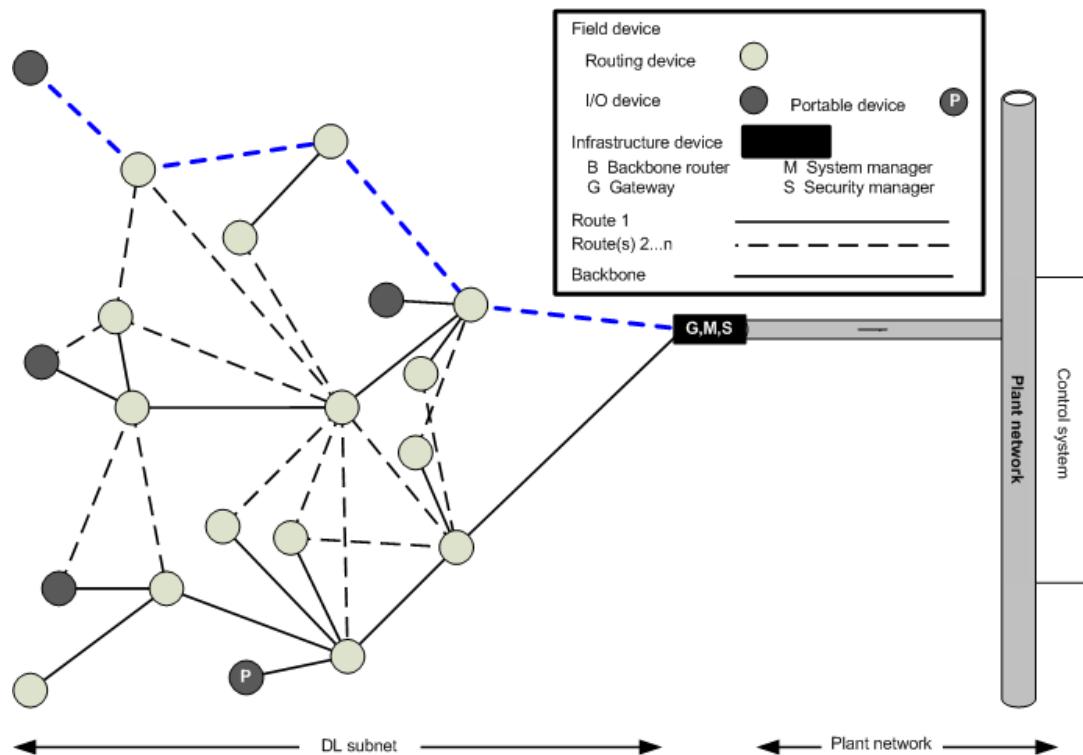


**Figure 97 – Delivery of an incoming NPDPU at its final destination**

### 10.2.7 Routing examples

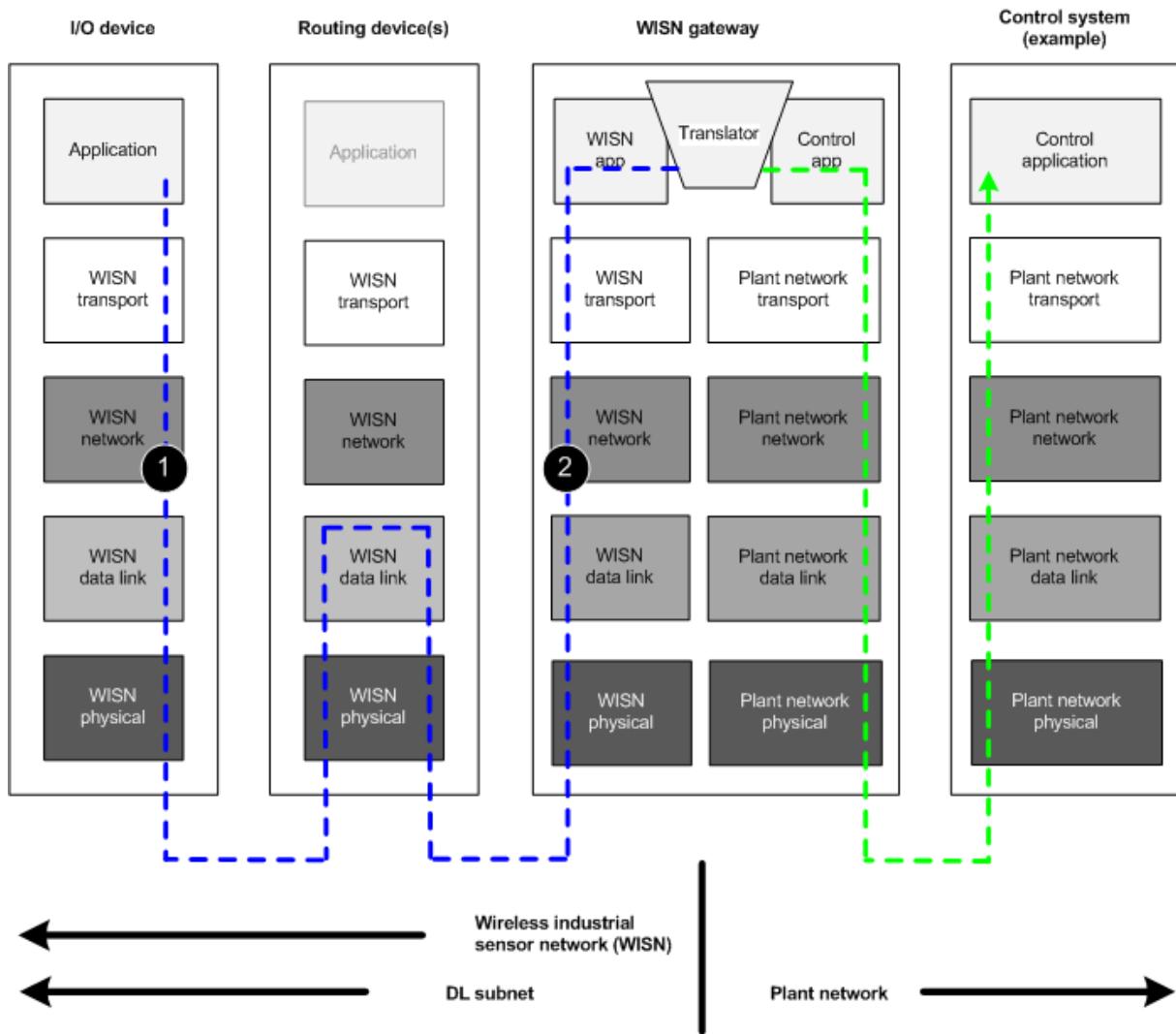
#### 10.2.7.1 Routing from a field device to a gateway on field

Figure 98 illustrates routing from a field device to a gateway with no backbone routing.



**Figure 98 – Routing from a field device to a gateway on field – no backbone routing**

Figure 99 depicts the flow through the communication protocol suites as the PDU moves from an I/O device to the gateway. It is assumed that the NPDPU needs no fragmentation.



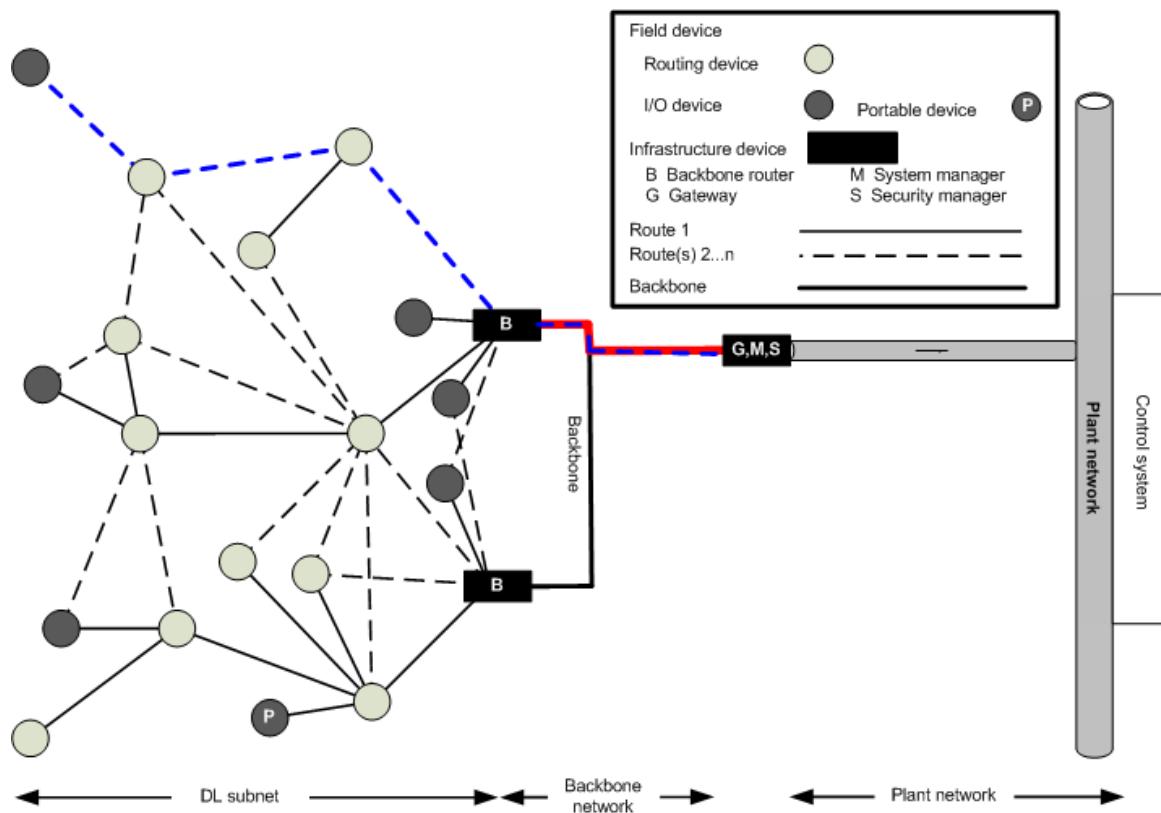
**Figure 99 – Protocol suite diagram for routing from a field device to a gateway on field – no backbone routing**

In Figure 99, the gateway is shown to have a field medium; hence no backbone network is involved in this example. The operations of the network layers at the devices that the PDU traverses (numbered in order) are as follows:

1. The network layer in the originating field device uses a basic network header; the 16-bit DL address of the gateway and the 16-bit DL address of the device itself are obtained from the ATT and passed to the DL through DD-DATA.request. The DL layer sends the PDU to the gateway.
2. The network layer in the gateway receives the NPDU, checks that the NPDU is intended for the gateway, translates the 16-bit DL address of the originating device (provided in DD-DATA.indication) into the 128-bit network address, and then passes the NSDU to the transport layer.

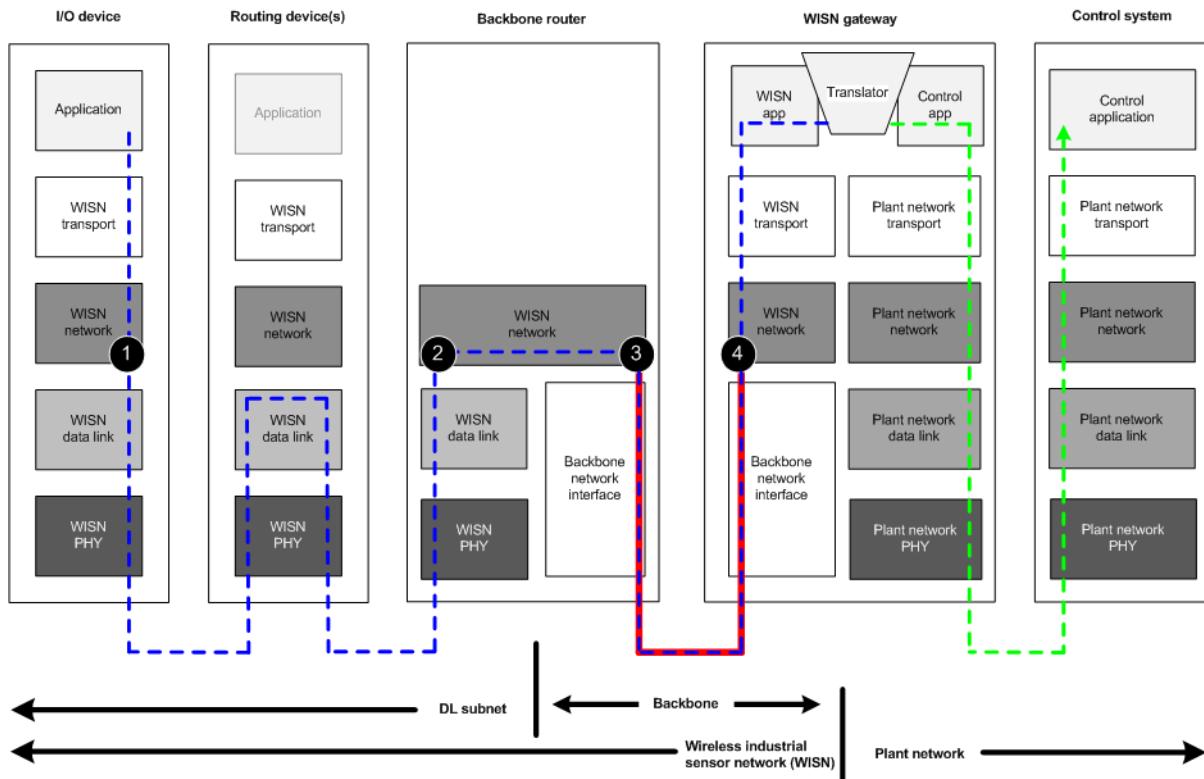
#### 10.2.7.2 Routing from a field device to a gateway on the backbone

Figure 100 illustrates the routing of a PDU from a field device to a gateway via a backbone router.



**Figure 100 – Routing a PDU from a field device to a gateway via a backbone router**

Figure 101 depicts the flow of a PDU from a field device to a gateway resident on the backbone network.



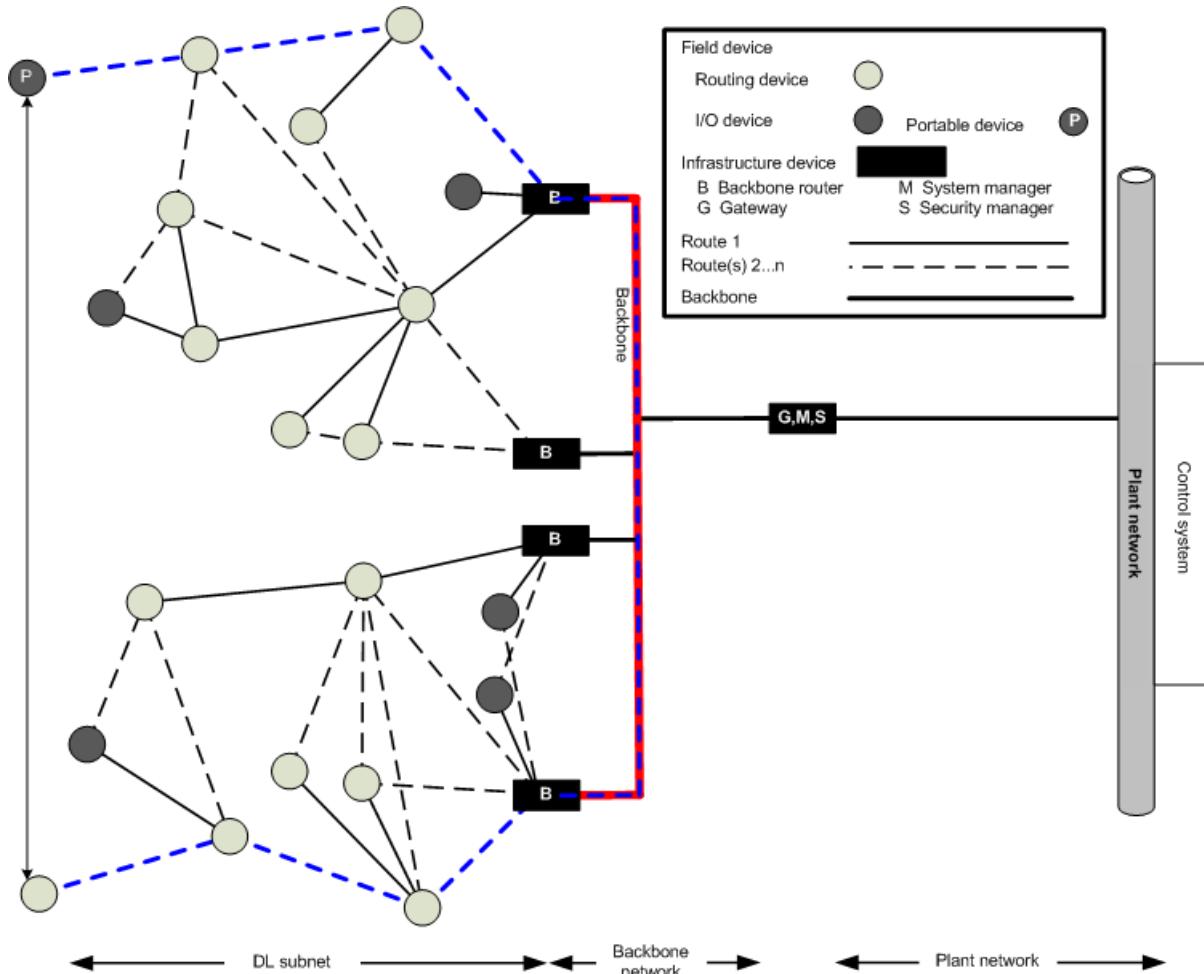
**Figure 101 – Protocol suite diagram for routing a PDU from a field device to a gateway via a backbone router**

The PDU is first routed to a backbone router over the DL subnet and from there to the gateway over the backbone. The operations in the network layers at the devices that the PDU traverses (numbered in order) are as follows:

1. The network layer of the I/O device passes to the DL its own 16-bit DL address as the source address and the 16-bit DL address of the gateway as the final destination address. If the ContractTable indicates that the ContractID needs to be included in the NPDU, the contract-enabled header is used; otherwise, the basic header is used if the compression used by the transport allows it (see 10.5.2.1). If the size of the NPDU is larger than maxDSDU size, the NPDU is fragmented. The PDU (or the set of fragments) is then handed to the DL.
2. The DL sends the PDU (or set of fragments) to the backbone router. If fragmented in (1), the set of fragments are reassembled to the NPDU at the backbone router. The network layer at the backbone router receives the NPDU and determines that the NPDU is not intended for the backbone router, since the final destination address in the DD-DATA.indication is the 16-bit DL address of the gateway. The backbone router translates this 16-bit DL address into the 128-bit network address of the desired gateway, uses its routing table to determine the next-hop address to reach the gateway and creates a full header (format shown in Table 216).
3. The NPDU with the expanded network header is presented to the backbone interface. The backbone interface routes the NPDU towards its final destination. In this example, the next hop is the final destination (the gateway).
4. The NPDU arrives at the network layer of the gateway over the backbone. The network layer at the gateway determines that the final destination address is equal to the address of the gateway itself and passes the NSDU to the transport layer.

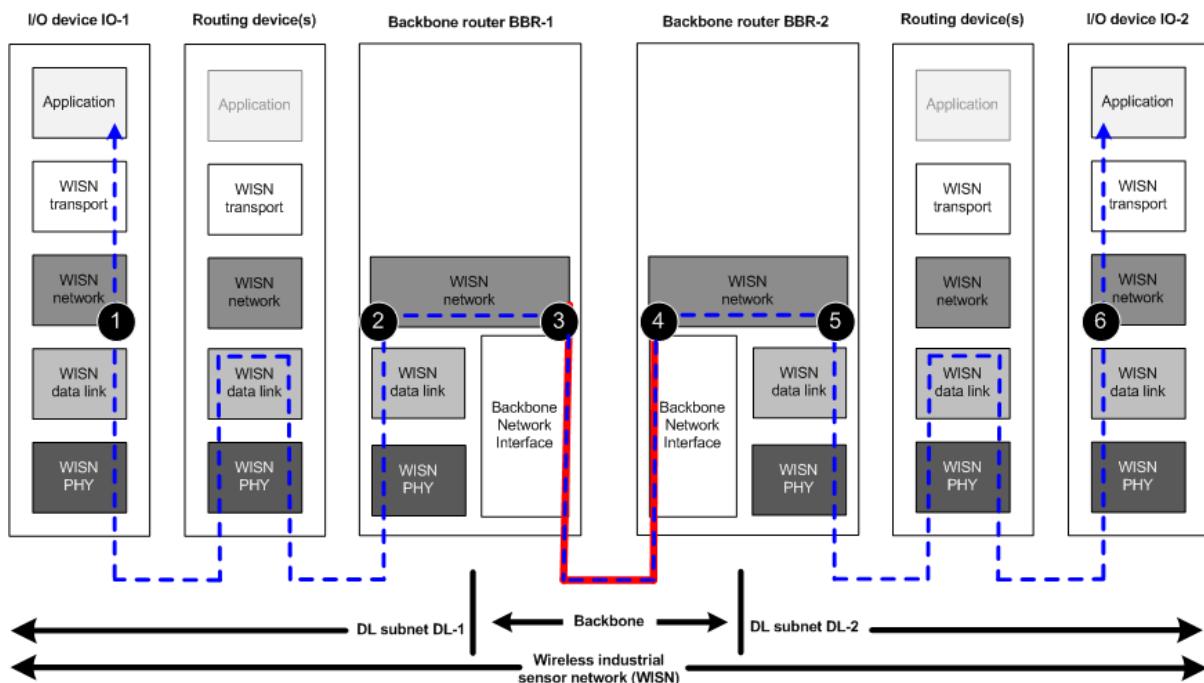
#### **10.2.7.3 Routing from a field device to another field device on a different DL subnet**

Figure 102 illustrates routing from an I/O device on one subnet to another I/O device on a different subnet.



**Figure 102 – Routing from a field device on one subnet to another field device on a different subnet**

Figure 103 shows the flow of a PDU between two field devices on different DL subnets. It is assumed that the NPDU needs no fragmentation.



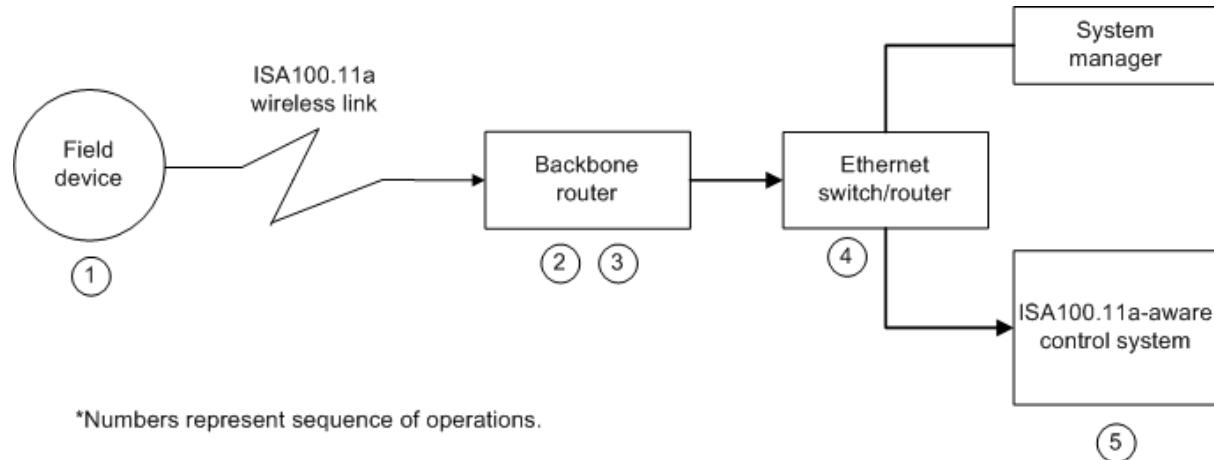
**Figure 103 – Protocol suite diagram for routing from an I/O device on one subnet to another I/O device on a different subnet**

The PDU is routed over the backbone from one DL subnet to the other. The operations performed by the network layers of the devices as the PDU flows from the originating I/O device (I/O-1) located in DL subnet DL-1 to the destination I/O device (I/O-2) located in DL subnet DL-2 are as follows:

1. The network layer at I/O-1 creates the NPDU using the contract-enabled network header. The network layer passes the NPDU to DL along with the 16-bit DL address of I/O-1 within DL-1 as the source address, and the 16-bit DL address of I/O-2 in DL-1 as the final destination address in DD-DATA.request. The ContractID is placed in the FlowLabel field of the contract-enabled header.
2. The PDU is routed over DL-1 and arrives at the network layer of BBR-1, i.e., the backbone router in DL-1. The NPDU is checked to see if it is destined for BBR-1 itself. Since it is not destined for BBR-1, the 16-bit DL addresses of I/O-1 and I/O-2 in the DD-DATA.indication are translated into their 128-bit network addresses.
3. The expanded header (in the format defined in Table 216) is created and presented to the backbone interface. The next hop for this NPDU over the backbone is determined by looking up the 128-bit network address of the final destination in the RT. The RT returns the 128-bit network address of BBR-2 (the backbone router of DL-2) as the next hop, since BBR-2 is the backbone router serving I/O-2. The ContractID is placed in the FlowLabel field of the expanded header. The priority of the PDU is placed in the TrafficClass field.
4. The network layer at BBR-2 receives the NPDU over the backbone. The NPDU indicates that the final destination is the 128-bit network address of I/O-2. This NPDU is then prepared for routing over DL-2 to reach I/O-2.
5. The network layer at BBR-2 creates a basic header. In the subsequent DD-DATA.request, the 16-bit DL address of I/O-1 in DL-2 is the originator address and the 16-bit DL address of I/O-2 in DL-2 is the final destination. The ContractID, extracted from the FlowLabel field of the expanded header, and the priority, extracted from the TrafficClass, are also passed down to the DL layer to enable selection of the appropriate routing resources (GraphID, etc.).
6. The NPDU arrives at the network layer of I/O-2. Since the final destination indicated in the DD-DATA.indication is the 16-bit DL address of I/O-2 in DL-2, the network layer translates the addresses into 128-bit and passes the NSDU to the transport layer of I/O-2.

#### 10.2.7.4 Ethernet backbone routing example

Figure 104 is an example of an implementation of the protocol suite diagram illustrated in Figure 101. In this network, a field device communicates with a control system that is aware of the native protocol of this standard; the backbone network in this example is an Ethernet network carrying IPv6 traffic.



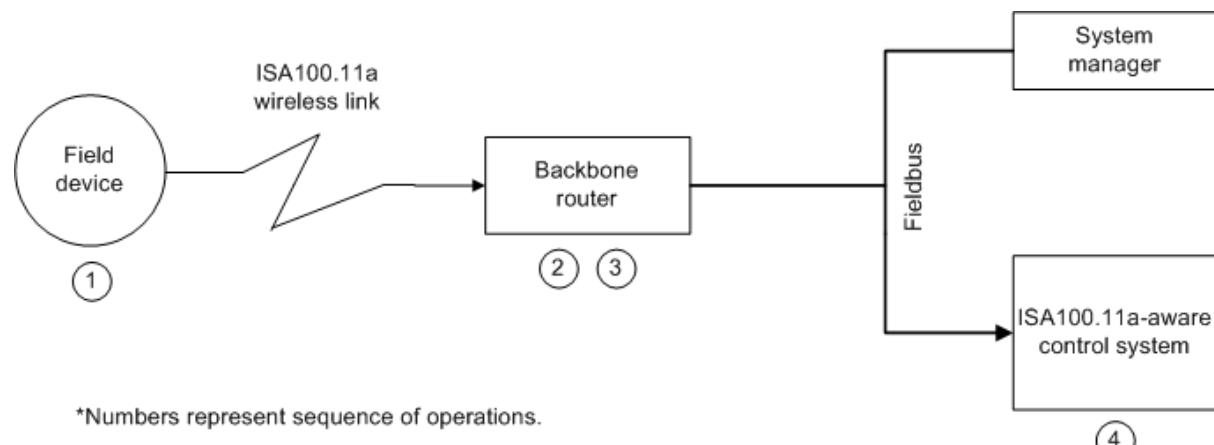
**Figure 104 – Routing over an Ethernet backbone network**

The numbered circles in Figure 104 indicate steps in the routing process and where they occur:

1. The network layer at the field device creates a NPDU and hands it to the DL for transmission to the next network layer hop (backbone router). The final destination address of the DPDU (provided by the network layer in DD-DATA.request) is the 16-bit DL address for the native protocol-aware control system. The DL mesh delivers the NPDU to the network layer of the backbone router.
2. The backbone router receives the NPDU and translates the 16-bit addresses into the 128-bit network addresses and expands the NPDU into a full header NPDU.
3. The backbone router sends the IPv6 NPDU over the Ethernet interface.
4. The Ethernet switch/router routes the Ethernet frame containing the NPDU to the control system.
5. The network layer at the control system receives the IPv6 NPDU from its Ethernet interface, performs network layer processing and passes the NSDU to the transport layer.

#### 10.2.7.5 Fieldbus backbone routing example

Figure 105 is a variant of Figure 104 that substitutes a generic fieldbus for the Ethernet backbone network.



**Figure 105 – Routing over a fieldbus backbone network**

The numbered circles in Figure 105 indicate steps in the routing process and where they occur:

1. The network layer at the field device creates a NPDU and hands it to the DL for transmission to the next network layer hop (backbone router). The final destination address of the DPDU is the 16-bit DL address for the native protocol-aware control system. The DL mesh delivers the NPDU to the network layer of the backbone router.
2. The backbone router receives the NPDU and translates the 16-bit address provided in DD-DATA.indication into the 128-bit network address and expands the NPDU into a full header NPDU.
3. The backbone router encapsulates the entire NPDU in one or more fieldbus frames addressed to the control system.
4. The control system (gateway) receives the fieldbus frame(s), extracts the NPDU, performs NL processing, and delivers the NSDU to the transport layer.

## 10.3 Network layer data services

### 10.3.1 General

The transport layer uses the interface supplied by the NDSAP to transmit and receive data. This interface is internal to a compliant device. The internal service access points (SAPs) of the device are depicted in Figure 16.

### 10.3.2 N-DATA.request

#### 10.3.2.1 General

N-DATA.request instructs the network layer to transmit data.

#### 10.3.2.2 Semantics

N-DATA.request (

```
DestAddress,  
ContractID,  
Priority,  
DE,  
NSDU,  
NSDULength,  
NSDUHandle,  
ECN  
)
```

Table 203 specifies the elements for N-DATA.request.

**Table 203 – N-DATA.request elements**

<b>Standard data type name: N-DATA.request</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
DestAddress (the 128-bit address of the destination of the NSDU)	1	Type: Device address Valid value set: 128-bit address
ContractID (the contract ID associated with the resources to be used for transmitting this NPDU; this ID is passed through directly to the DL)	2	Type: Unsigned16 Valid value set: [0, $2^{16}-1$ ]; 0 indicates no contract
Priority (priority of the message within the contract; this priority shall be combined with the 2-bit contract priority by the network layer to make the priority field 4 bits before passing it to the DL)	3	Type: Unsigned2
DE (indicates whether the PDU is eligible for discard)	4	Type: Unsigned1
NSDU (the set of octets forming the NSDU to be transmitted by the network layer, including the transport headers)	5	Type: Set of octets
NSDULength (the number of octets in the NSDU to be transmitted)	6	Type: Unsigned16
NSDUHandle (the handle associated with the NSDU to be transmitted)	7	Type: Unsigned16
ECN (explicit congestion notification)	8	Type: Unsigned2

### 10.3.2.3 Appropriate usage

The transport layer invokes N-DATA.request to transmit an NSDU.

### 10.3.2.4 Effect on receipt

On receipt of N-DATA.request, the network layer constructs the network layer headers of the NPDU (eliding the first octet LoWPAN\_NHC of the NSDU if the basic header is used), fragments the NPDU (if necessary), and conveys it to the lower layers for transmission. If ContractID is 0, the PDU is recognized as a join-related message and the EUI-64 of the destination is derived from the link-local address passed in DestAddress.

### 10.3.3 N-DATA.confirm

#### 10.3.3.1 General

N-DATA.confirm reports the result of an N-DATA.request.

#### 10.3.3.2 Semantics

The semantics of N-DATA.confirm are as follows:

```
N-Data.confirm ( NSDUHandle,  
                  status  
                )
```

Table 204 specifies the elements for N-DATA.confirm.

**Table 204 – N-DATA.confirm elements**

Element name	Element identifier	Element scalar type
NSDUHandle (the handle of the NSDU whose status is being reported)	1	Type: Unsigned16
Status (the result of a N-DATA.request primitive)	2	Type: Enumeration Valid value set: SUCCESS, FAILURE

#### 10.3.3.3 When generated

The network layer generates N-DATA.confirm in response to an N-DATA.request. N-DATA.confirm returns a status value that indicates either SUCCESS or FAILURE.

#### 10.3.3.4 Appropriate usage

On receipt of N-DATA.confirm, the transport layer is notified of the result of its request to transmit an NSDU.

### 10.3.4 N-DATA.indication

#### 10.3.4.1 General

N-DATA.indication signals to the transport layer that the network layer received an NSDU destined for the transport layer.

#### 10.3.4.2 Semantics

The semantics of N-DATA.indication are as follows:

```
N-Data.indication ( SrcAddress,  
                     NSDU,  
                     NSDULength,  
                     ECN,  
                     Priority  
                   )
```

Table 205 specifies the elements for N-DATA.indication.

**Table 205 – N-DATA.indication elements**

<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
SrcAddress (the 128-bit address of the source of the NSDU)	1	Type: Device address Valid value set: 128-bit address
DestAddress (the 128-bit address of the destination of the NSDU i.e. the device's own 128 bit address.)	2	Type: Device address Valid value set: 128-bit address
NSDU (the set of octets forming the received NSDU, including the transport layer headers)	3	Type: Set of octets
NSDULength (the number of octets in the received NSDU)	4	Type : Unsigned16
ECN (explicit congestion notification)	5	Type: Integer
Priority (4-bit PDU priority as received)	6	Type: Unsigned4

#### **10.3.4.3 Appropriate usage**

The network layer invokes N-DATA.indication to notify the transport layer of a received NPDU with the NSDU destined for the transport layer. If the NPDU received contains the basic header, then, before passing the NSDU to the transport, the network layer shall restore (prepend) the first octet of the NSDU LoWPAN\_NHC = "11110111" (that had been elided when the basic header was constructed). If the source address received from the DL is an EUI-64, a 128-bit link-local address is constructed from the EUI-64 and passed in the SrcAddress field of the N-DATA.indication. The DestAddress in N-DATA.indication is the 128-bit link local address of the device for join-related messages and globally assigned 128-bit address for normal operation.

#### **10.3.4.4 Effect on receipt**

On receipt of N-DATA.indication, the transport layer shall begin processing the received NSDU.

### **10.4 Network layer management object**

#### **10.4.1 Network layer management information base**

Table 206 specifies the attributes of the network layer management object (NLMO).

**Table 206 – NLMO attributes**

<b>Standard object type name: NLMO (Network layer management object)</b>				
<b>Standard object type identifier: 123</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute type	Description of behavior of attribute
Backbone_Capable	1	A Boolean flag to indicate if the device is backbone capable or not	Type: Boolean	Fixed value based on device capabilities and implementation details. Backbone capability may be ignored by system manager
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: 0-1	
DL_Capable	2	A Boolean flag to indicate if the device is capable to communicate over the DL of this standard	Type: Boolean	Fixed value based on device capabilities and implementation details. DL interface capability may be ignored by the system manager
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: 0 – 1	
16-bit DL Address	3	The 16-bit DL address of the device	Type: Unsigned16	A fixed value as assigned by the system manager at join. The attribute shall remain same during normal operation. Write access is provided to the system manager only. Assignment of a new address shall trigger a Join_Command = 3, restart as provisioned following which the device shall have to re-join the network. This attribute is a duplicate of the corresponding attributes in the DMO and DLMO. This attribute is a duplicate of the corresponding attributes in the DMO and DLMO
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: N/A	
			Valid value set: [1 to 2 <sup>15</sup> -1]	
Long_Address	4	The 128-bit address of the device	Type: Unsigned128	A fixed value as assigned by the system manager at join. The attribute shall remain same during normal operation. Write access is provided to the system manager only. Assignment of a new address shall trigger a Join_Command = 3, restart as provisioned following which the device shall have to re-join the network. This attribute is a duplicate of the corresponding attributes in the DMO and DLMO
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: N/A	
			Valid value set: [ 0 – 2 <sup>128</sup> -1]	
Route_Table	5	The routing table that includes information to route a PDU.	Type: Array of NLRouteTbl structures (see NLRouteTbl)	The Routing table consists of a destination address, next network hop for that destination and the number of network hops needed. The Routing table structure, its corresponding alerts and methods are discussed below. The size of the routing table present in devices that only operate within the DL subnet and are not backbone capable is presented in Table 495
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: N/A	
			Valid value set: See Table 208	
Enable_Default_Route	6	A Boolean value set to indicate if a default routing is enabled	Type: Boolean	Enables a default route
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Disabled	

<b>Standard object type name: NLMO (Network layer management object)</b>				
<b>Standard object type identifier: 123</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute type</b>	<b>Description of behavior of attribute</b>
			Valid value set: [0 - 1]	
Default_Route_Entry	7	Destination address associated with the default route	Type : Unsigned128 Classification: Static Accessibility: Read/write Initial default value: N/A  Valid value set: [ 0 – 2 <sup>128</sup> -1]	Can be used to look up the default route in the route table. Packets that include a destination address with no corresponding entry in the routing table shall be forwarded utilizing the default route. Field devices may utilize the default route to send packets to devices operating on the backbone that have not been assigned 16-bit addresses, but only 128-bit addresses
Contract_Table	8	Includes information to construct the network header for a particular PDU flow	Type: Array of NLContractTbl structures (see NLContractTbl)  Classification: Static  Accessibility: Read/write  Initial default value: N/A  Valid value set: See Table 207	The contract table consists of the destination address, priority for a particular flow. The table also includes information indicating if a ContractID needs to be carried in the NPDU or not. The scope of a contract ID is local to a device, however, the combination of a source address (128-bit) and ContractID is globally unique. The contract table structure and its corresponding alerts and methods are discussed below
Address_Translation_Table	9	Includes the 128-bit address and the 16-bit alias for the 128-bit addresses	Type: Array of NLATTbl structures (see NLATTbl)  Classification: Static  Accessibility: Read/write  Initial default value: N/A  Valid value set: See Table 209	The address translation table in a device is indexed by the 128-bit long address and stores the corresponding 16-bit alias for that long address in the DL subnet to which the device belongs
Max_NSDU_size	10	Maximum service data unit size supported by the network layer of the device	Type: Unsigned16  Classification: Constant  Accessibility: Read only  Initial default value: 70  Valid value set: 70-1280	Fixed value based on device memory capabilities and implementation details. The value of 1280 comes from IETF RFC 4944. See 6.2.8 on how this value can control fragmentation at the application layer. NSDUs that exceed the maximum value may be rejected by the device
Frag_Reassembly_Timeout	11	Amount of time (in seconds) a reassembly buffer needs to be held open for fragments to come in before discarding the NPDU	Type: Unsigned16  Classification: Static  Accessibility: Read/write  Initial default value: 60  Valid value set: 1-600	The default value is 60 s, but the system manager can change this value
Frag_Datagram_Tag	12	Current tag number for fragmentation at the device	Type: Unsigned16  Classification: Dynamic  Accessibility: Read only  Initial default value: Random  Valid value set: [0 – 65535]	A new tag number is used for every NPDU that needs to be fragmented. The Tag number is incremented by one for each NPDU to be fragmented. Value wraps back to zero after 65535

<b>Standard object type name: NLMO (Network layer management object)</b>				
<b>Standard object type identifier: 123</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute type	Description of behavior of attribute
NLRouteTblMeta	13	Metadata for Route Table attribute (Attribute 5)	Type: Meta_Data_Attribute  Classification: Static  Accessibility: Read only  Initial default value: N/A  Valid value set: N/A	Metadata containing a count of the number of entries in the table and capacity (the total number of rows allowed) for this table. The size of the routing table present in devices that only operate within the DL subnet and are not backbone capable is presented in Table 495
NLContractTblMeta	14	Metadata for Contract Table attribute (Attribute 8)	Type: Meta_Data_Attribute  Classification: Static  Accessibility: Read only  Initial default value: N/A  Valid value set: N/A	Metadata containing a count of the number of entries in the table and capacity (the total number of rows allowed) for this table.
NLATTtblMeta	15	Metadata for Address Translation Table attribute (attribute 9)	Type: Meta_Data_Attribute  Classification: Static  Accessibility: Read only  Initial default value: N/A  Valid value set: N/A	Metadata containing a count of the number of entries in the table and capacity (the total number of rows allowed) for this table.
DroppedNPDU AlertDescriptor	16	Describes how a dropped NPDU alert is reported on the network	Type : Alert report descriptor  Classification: Static  Accessibility: Read/write  Initial default value: Alert report disabled = True Alert report priority = 7  Valid value set: See type definition	

#### 10.4.2 Structured management information bases

The NLMO defines three structured management information bases (SMIBs) as tables. They are the contract table, the route table (RT), and the address translation table (ATT).

Table 207 specifies the elements for the contract table. Devices that are not backbone capable may elide the Source\_Address field.

**Table 207 – Contract table structure**

<b>Standard data type name: NLContractTbl</b>		
<b>Standard data type code: 441</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
Contract_ID*	1	Type: Unsigned 16 Size = 2 octets Classification: Static Accessibility: Read/write Default value : N/A Valid value set: 0 – 65535
Source_Address*	2	Type: Unsigned 128 Size = 16 octets Classification: Static Accessibility: Read/write Default value : 0 Valid value set: [ 0 – 2 <sup>128</sup> -1]
Destination_Address	3	Type: Unsigned 128 Size = 16 octets Classification: Static Accessibility: Read/write Default value : 0 Valid value set: [ 0 – 2 <sup>128</sup> -1]
Contract_Priority	4	Type: Unsigned 2 Size = 2 bits Classification: Static Accessibility: Read/write Default value : 00 Valid value set: [00,01,10,11]
Include_Contract_Flag	5	Type: Boolean Size = 1bit Classification: Static Accessibility: Read/write Default value : 0 Valid value set: [0=Do not include, 1=Include]

\* indicates an index field

Table 208 specifies the elements for the route table.

**Table 208 – Route table elements**

<b>Standard data type name: NLRouteTbl</b>		
<b>Standard data type code: 442</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
Destination_Address*	1	Type: Unsigned 128 Size = 16 octets Classification: Static Accessibility: Read/write Default value : N/A Valid value set: [ 0 – 2 <sup>128</sup> -1]
Next_Hop	2	Type: Unsigned 128 Size = 16 octets Classification: Static Accessibility: Read/write Default value : N/A Valid value set: [ 0 – 2 <sup>128</sup> -1]
NWK_HopLimit	3	Type: Unsigned 8 Size =1 octet Classification: Static Accessibility: Read/write Default value : 64 Valid value set: [ 0 – 255]
Outgoing_Interface	4	Type: Boolean Size = 1bit Classification: Static Accessibility: Read/write Default value : 0 Valid value set:[0=DL, 1=Backbone]

\* indicates an index field

Table 209 specifies the elements for the address translation table.

**Table 209 – Address translation table structure**

<b>Standard data type name: NLATTbl</b>		
<b>Standard data type code: 443</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
Long_Address*	1	Type: Unsigned 128 Size = 16 octets Classification: Static Accessibility: Read/write Default value : N/A Valid value set: [ 0 – 2 <sup>128</sup> -1]
Short_Address	2	Type: Unsigned 16 Size = 2 octets Classification: Static Accessibility: Read/write Default value : N/A Valid value set: [ 0 – 32767]

\* indicates an index field

#### 10.4.3 Network layer management object methods

Standard methods such as read and write can be used for scalar or structured MIBs in their entirety. The methods described in this subclause are used to manipulate tables. The methods in this subclause allow access to a particular row of a structured MIB based on a unique index field. All devices shall be capable of manipulating the NLMO attributes immediately. Device support delayed manipulation of these attributes utilizing cutover methods but are not required to do so.

It is assumed that the tables have a unique index field, which may either be a single element or the concatenation of multiple elements. The index field is assumed to be the (concatenation

of the) first (few) element(s) of the table. For example, the contract table index field is the concatenation of the Contract\_ID and Source\_Address.

These methods do not specify the interface between the NME and NLMO, as this interface is internal to a particular device. The management entity may keep local copies of the MIBs.

Table 210 describes the methods for manipulation of structured MIBs. These methods are based on the Read\_Row, Write\_Row, and Delete\_Row templates defined in Annex J.

**Table 210 – NLMO structured MIB manipulation methods**

<b>Standard object type name: NLMO</b>		
<b>Standard object type identifier: 123</b>		
<b>Defining organization: ISA</b>		
Method name	Method ID	Method description
Set_row_RT	1	Method to set (either add or edit) the value of a single row of the route table. The method uses the Write_Row method template defined in Annex J with the following arguments: Attribute_ID :5 (route table) Index 1: 1 (Destination_address)
Get_row_RT	2	Method to get the value of a single row of the route table. The method uses the Read_Row method template defined in Annex J with the following arguments: Attribute_ID :5 (route table) Index 1: 1 (Destination_address)
Delete_row_RT	3	Method to delete a single row of the contract table. The method uses the Delete_Row method template defined in Annex J with the following arguments: Attribute_ID :5 (route table) Index 1: 1 (Destination_address)
Set_row_ContractTable	4	Method to set (either write or edit) the value of a single row of the contract table. The method uses the Write_Row method template defined in Annex J with the following arguments: Attribute_ID :8 (contract table) Index 1: 1 (ContractID) Index 2: 2 (Source Address)
Get_row_ContractTable	5	Method to get the value of a single row of the contract table. The method uses the Read_Row method template defined in Annex J with the following arguments: Attribute_ID :8 (contract table) Index 1: 1 (ContractID) Index 2: 2 (Source Address)
Delete_row_ContractTable	6	Method to delete the value of a single row of the contract table. The method uses the Delete_Row method template defined in Annex J with the following arguments: Attribute_ID :8 (contract table) Index 1: 1 (ContractID) Index 2: 2 (Source Address)
Set_row_ATT	7	Method to set (either add or edit) the value of a single row of the Address Translation table. The method uses the Write_Row method template defined in Annex J with the following arguments: Attribute_ID :9 (AT table) Index 1: 1 (Long Address)
Get_row_ATT	8	Method to get the value of a single row of the Address Translation table. The method uses the Read_Row method template defined in Annex J with the following arguments: Attribute_ID :9 (AT table) Index 1: 1 (Long Address)
Delete_row_ATT	9	Method to delete a single row of the Address Translation table. The method uses the Delete_Row method template defined in Annex J with the following arguments: Attribute_ID :9 (AT table) Index 1: 1 (Long Address)

Table 211 describes the alert to indicate a dropped PDU/PDU error.

**Table 211 – Alert to indicate dropped PDU/PDU error**

<b>Standard object type name(s): NLMO</b>					
<b>Standard object type identifier: 123</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Alert to indicate dropped PDU /PDU error</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority (Enumerated: high, med, low, journal only)	Value data type	Description of value included with alert
0 = Event	1 = Comm. diagnostic	0 = NL_Dropped_PDU	7 =Medium	Type: OctetString	<p>The Value is an octet string consisting of at least two octets. The first octet specifies the length of the value field included with the alert in octets.</p> <p>The second octet is an enumeration of the diagnostic code with the following values:</p> <ul style="list-style-type: none"> <li>1 – Destination unreachable</li> <li>2 – Fragmentation error</li> <li>3 – Reassembly timeout</li> <li>4 – Hop Limit reached</li> <li>5 – Header errors</li> <li>6 – Next hop unreachable (No route)</li> <li>7 – Out of memory</li> <li>8 – NPDU length too long</li> <li>9-255 – reserved</li> </ul> <p>The remaining octets include the NPDU header for the dropped PDU</p>

## 10.5 Network layer protocol data unit formats

### 10.5.1 General

Each NPDU shall consist of two basic components:

- A network header possibly comprising addressing, class of service (CoS), and fragmentation fields.
- A network payload of variable length containing the data that needs to be transmitted.

This standard shall allow three different NPDU header formats:

- The basic header;
- The contract-enabled header; and
- The full header.

Devices compliant with this standard shall use dispatch prefixes (the least significant 3 bits of the first byte of the NPDU) to distinguish between these header formats (see Figure 106). The prefix 000 shall indicate that the NPDU header format is the basic header. The basic header prefix conforms to the NALP dispatch of 6LoWPAN.

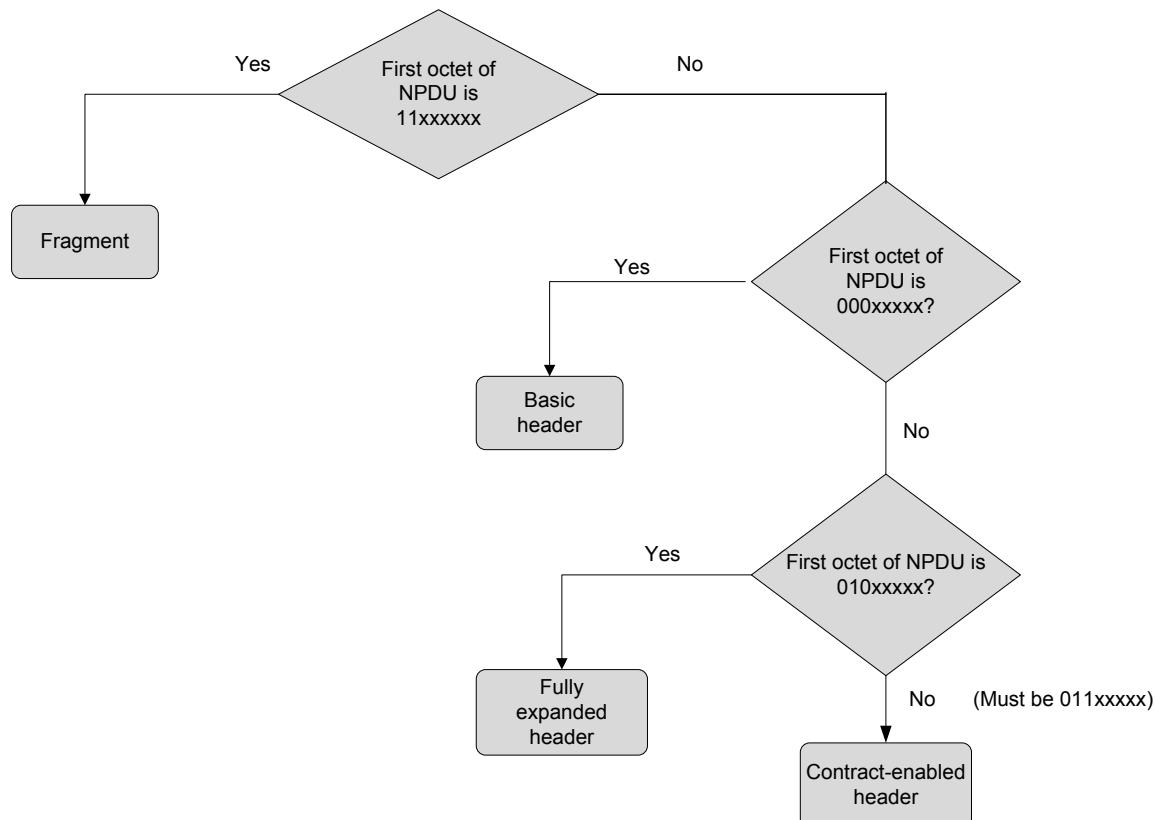
The prefix 011 shall indicate the contract-enabled header. The contract-enabled header conforms to the LOWPAN\_IPHC dispatch of 6LoWPAN.

The prefix 010 shall indicate the full header format. Resource constrained devices that operate in the wireless subnet and are backbone capable may construct the full header but are not required to do so. If a packet that is constructed utilizing the full header is received by

a device that is not backbone capable, the device may discard the NPDU and send a dropped PDU/PDU error alert with value 5 indicating a header error.

Finally, a prefix of 110 or 111 shall indicate that the PDU is a fragment of a larger NPDU that needs to be reassembled.

**NOTE** This standard primarily uses 16-bit addresses for very first (originator) and final destination in the DPDUs transmitted over the IEEE Std 802.15.4 wireless network. Therefore, the network header is recommended to be either the basic or contract-enabled header format for the NPDUs transmitted over the IEEE Std 802.15.4 wireless links. It is not recommended, although not prohibited, to use the full header format for transmission of NPDUs over the field medium.



**Figure 106 – Distinguishing between NPDU header formats**

The dispatch octet bit patterns are shown in Table 212.

**Table 212 – Common header patterns**

Dispatch pattern	Header type
000xxxxx	NALP (Not A LoWPAN frame) – Basic header
010xxxxx	IPv6 (uncompressed IPv6 header) – Full header
011xxxxx	LoWPAN_IPHC (compressed IPv6 header) – Contract-enabled header
11xxxxxx	LoWPAN fragment

The network layer headers shall follow the formats defined herein.

### 10.5.2 Basic header format for network layer

#### 10.5.2.1 Intended usage

The DL of this standard employs a link level mesh. In the most common case, a PDU will traverse a single DL subnet, so the basic header is optimized to minimize the NPDU overhead. The route that needs to be taken by the PDU is known to the device of ingress into the DL subnet; this device of ingress makes all the necessary DL routing decisions. The ContractID is not transmitted in the basic network header.

The basic header for the NL shall be used only if the user datagram protocol (UDP) header is fully compressed (i.e., the source and destination port numbers are compressed to four (4) bits each and the UDP checksum is elided). The network layer can determine whether the UDP header is fully compressed by looking at the LOWPAN\_NHC octet, which is always the first octet of the NSDU passed to the network layer by the transport layer. Since the basic header is used only in case of fully compressed UDP header (i.e., fixed and known value of UDP LOWPAN\_NHC) the UDP LOWPAN\_NHC octet shall be elided by the network layer of origin and restored by the destination network layer.

### 10.5.2.2 Format

Table 213 describes the basic network header format.

**Table 213 – Basic network layer header format**

<b>octets</b>	<b>bits</b>							
	7	6	5	4	3	2	1	0
1	Dispatch							
(variable)	Network payload							

The fields of the basic network layer header shall be as follows:

- Dispatch: The dispatch field indicates the network layer header format. For the basic header, the dispatch field shall have the value 00000001.

### 10.5.2.3 Relation to 6LoWPAN (INFORMATIVE)

The 6LoWPAN format allows all fields of the full IPv6 header to be elided. The dispatch and encoding octets to achieve this would be 01111110 01110111 , which indicates:

- Dispatch = 011
- TF = 11 (both traffic class and flow label elided)
- NH = 1 (next header field elided)
- HLIM = 10 (HopLimit = 64)
- CID = 0 (no context identifier extension)
- SAC = 1 (stateful source address compression; the ATT provides the context)
- SAM = 11 (source address fully elided)
- M = 0 (no multicast)
- DAC = 1 (stateful destination address compression; the ATT provides the context)
- DAM = 11 (destination address fully elided)

The 6LoWPAN format also allows the UDP header to be compressed so that the source and destination port numbers are four (4) bits each and the checksum is elided. In this case, the UDP LOWPAN\_NHC field has the value 11110111 , which indicates:

- Protocol = 11110 (UDP)
- Checksum compression = 1 (checksum elided)
- Port compression = 11 (source and destination ports are compressed to four (4) bits each and their implied prefix is 0xF0B)

The basic header is essentially a single-octet abbreviation for the three octets (two octets of fully compressed IP header and one octet of fully compressed UDP header) noted above. Since it is an abbreviation, it is fully compatible with the 6LoWPAN format. A device receiving a basic header NPDU can expand the basic header dispatch octet to the three octets noted above and obtain a 6LoWPAN-compliant PDU.

### 10.5.3 Contract-enabled network header format

#### 10.5.3.1 Intended usage

Like the basic header, the contract-enabled network header is intended for use within DL subnets. The very first (originator) device of an NPDU may use the contract-enabled header instead of the basic header if it is desirable for devices other than the very first (originator) device to be aware of the NPDU stream to which the NPDU belongs. For example, an intermediate router on the backbone may need to select:

- Appropriate backbone resources upon egress from the originating DL subnet; or
- Appropriate DL resources (graph ID, priority, etc.) upon ingress into a destination DL subnet.

The contract includes a flag to indicate to the originating network layer whether the network header requires inclusion of the ContractID.

The contract-enabled header shall also be used if the UDP LOWPAN\_NHC does not indicate full compression of the UDP header. Join process messages between the new device and the advertising router will always fall under this category since they do not elide the UDP checksum.

#### 10.5.3.2 Format

Table 214 describes the contract-enabled header format.

**Table 214 – Contract-enabled network layer header format**

octets	bits												
	7	6	5	4	3	2	1	0					
1	LOWPAN_IPHC dispatch			LOWPAN_IPHC encoding (bits 8-12)									
2	LOWPAN_IPHC encoding (bits 0-7)												
3 (opt)	Octet alignment			FlowLabel (bits16-19)									
4 (opt)	Flow Label (bits 8-15)												
5 (opt)	Flow Label (bits 0-7)												
6 (opt)	HopLimit												
(variable)	Network payload												

Fields include:

- LOWPAN\_IPHC dispatch: This field shall indicate that the header format is contract-enabled and that LOWPAN\_IPHC header compression encoding bits follow. The LOWPAN\_IPHC dispatch field shall be 011.
- LOWPAN\_IPHC encoding: This field is 13 bits long and its value shall be 011HH01110111 (if the optional octets 3 through 5 are present to carry the contract ID) or 111HH01110111 (if the optional octets 3 through 5 are elided); in either case, HH shall have the value 00 if HopLimit is carried in line, 01 if the hop limit is 1, 10 if the hop limit is 64 and 11 if the hop limit is 255.
- FlowLabel: The lower order 16 bits of the FlowLabel shall be set to ContractID. The higher order 4 bits shall be all zeros. This field shall only be present if octets 3 through 5 are present, as indicated by LOWPAN\_IPHC encoding.
- HopLimit: This field shall indicate the number of Layer 3 hops permitted before the NPDU is discarded. The HopLimit field shall be set to a value indicated by the device's routing table (RT; see 10.2.6.2). The default value for HopLimit field shall be 64. Devices that only operate within the DL subnet and are not backbone capable shall set the HopLimit to 1. From the perspective of the network layer, any device reachable through the DL mesh is a single network hop away.

For join process messages, LOWPAN\_IPHC encoding shall have the value 1110101110111 to indicate that octets 3 through 5 are elided (no contract ID) and hop limit is 1 (HopLimit field elided).

### 10.5.3.3 Relation to 6LoWPAN (INFORMATIVE)

Table 215 shows the 6LoWPAN\_IPHC encoding format.

**Table 215 – 6LoWPAN\_IPHC encoding format**

octets	bits							
	7	6	5	4	3	2	1	0
1	0	1	1	TF		NH	HLIM	
2	CID	SAC	SAM		M	DAC	DAM	
3	Non-compressed fields							
...								
N								

The encoding for the contract-enabled network layer header is derived by using the following values for the 6LoWPAN\_IPHC encoding fields:

- TF = 01 or 11. In a 6LoWPAN format, TF = 01 implies a 3-octet inline field is carried in the non-compressed fields. This inline field consists of 2 bits of ECN, followed by a 2-bit pad, followed by 20 bits of flow label. Since in this standard the ECN is always enabled and the congestion indication is carried by the lower layer, both the ECN and the 2-bit pad shall be all 0s. In a 6LoWPAN format, TF = 11 implies this 3-octet field is elided.
- NextHeaderNH = 1 to indicate that the next header can be inferred from the prefix of the transport header. In this standard, the next header is always UDP.
- HLIM = HH. These bits are used by this standard to indicate the hop limit compression scheme as intended in 6LoWPAN.
- CID = 0 to indicate no additional 8-bit context identifier extension is used.
- SAC = 1 to indicate stateful compression for the source address.
- SAM = 11 to indicate that all 128 bits of the source address are elided (since, in this standard, the 16-bit alias is carried by the lower layer and is indexed in the ATT, which provides the translation context).
- M = 0 to indicate the destination address is not multicast.
- DAC = 1 to indicate stateful compression for the destination address.
- DAM = 11 to indicate that all 128 bits of the source address are elided (since, in this standard, the 16-bit alias is carried by the lower layer and is indexed in the ATT, which provides the translation context).

### 10.5.4 Full header (IPv6) format

#### 10.5.4.1 Intended usage

The full header format is used primarily over the backbone network. A field device may also use the full header format instead of the basic or contract-enabled format but it is not recommended.

When the NPDU reaches an intermediate backbone router over the DL, the backbone router shall fully expand the header to include all fields as defined in the IPv6 header. It is necessary to expand the network layer addresses of the very first (originator) device and the final destination device to 128 bits to disambiguate their 16-bit DL addresses when routing outside the DL subnet.

#### 10.5.4.2 Format

Table 216 describes the IPv6 header format.

**Table 216 – IPv6 network layer header format**

octets	bits							
	7	6	5	4	3	2	1	0
1	Version						TrafficClass (bits 7-4)	
2	TrafficClass (bits 3-0)						FlowLabel (bits 19-16)	
3	FlowLabel (bits 15-8)							
4	FlowLabel (bits 7-0)							
5	PayloadLength (bits 15-8)							
6	PayloadLength (bits 7-0)							
7	NextHeader							
8	HopLimit							
9 -24	Destination address							
25-40	Source address							
(variable)	Network payload							

Fields include:

- Version: 4-bit IP version number shall be set to 6.
- TrafficClass: The higher order four bits of this 8-bit field shall be used to carry the 4-bit priority of the NPDU over the backbone. The fifth bit shall be 0 and the sixth bit shall be used to carry the Discard Eligible (DE) bit of the NPDU over the backbone. The two lowest-order bits shall carry the ECN.
- FlowLabel: The value of this field shall be as defined in the contract-enabled header format (see 10.5.3.2). This field shall be set to all zeros if the Contract ID is not carried as part of the flow.
- PayloadLength: This 16-bit unsigned integer shall contain the length of the IPv6 payload, i.e., the rest of the NPDU following this header, in octets.
- NextHeader: This 8-bit field shall be set to 00010001 (17 decimal) identifying the following header as UDP.
- HopLimit: The value of this 8-bit field shall be set to the number of network layer (Layer 3) hops needed to get to the destination. If the NPDU is received over a DL subnet with the basic header, this field shall be updated upon egress from the DL subnet by a backbone router according to the routing table of the router. This field is decremented by 1 by the network layer of each device if the network layer forwards the NPDU. The NPDU shall be discarded if HopLimit is decremented to zero.
- Destination address: This field shall contain the 128-bit network address of the final destination (intended recipient) of the NPDU.
- Source address: This field shall contain the 128-bit network address of the originator of the NPDU.

#### 10.5.4.3 Relation to 6LoWPAN (INFORMATIVE)

It is not recommended to use the full network layer header format in an NPDU being transmitted over the DL network of this standard. However, the standard does not preclude the use of the full header over DL subnets. If used over the DL network, the NPDU shall contain the IPv6 dispatch as shown in Table 217.

**Table 217 – Full network layer header in the DL**

### 10.5.5 Fragmentation header format

#### **10.5.5.1 Intended usage**

If an NPDU with length greater than the dlmo.maxDSDUSize needs to be transmitted over the DL, the NPDU shall be fragmented. When an NPDU needs to be fragmented, the fragmentation header shall be inserted.

### **10.5.5.2 Format**

The fragmentation header contains a 5-bit fragmentation type, followed by a 27-bit header for the first fragment and a 35-bit header for subsequent fragments. Fields of the basic, contract-enabled, or full headers from the dispatch octet onwards shall be placed in the first fragment only. Table 218 shows the network layer header format for fragmented NPDUs.

**Table 218 – Network layer header format for fragmented NPDUs**

## Fields include:

- Fragmentation type: This 5-bit field shall be set to 11000 for the first fragment and to 11100 for the second and subsequent fragments.
  - Fragmentation header:
  - First fragment: The first fragment shall contain the first fragment header as defined below.

**Table 219 – First fragment header format**

- Subsequent fragments: The second and subsequent fragments (up to and including the last) shall contain a fragmentation header shown below.

**Table 220 – Second and subsequent fragment header format**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Fragmentation type								
	1	1	1	0	0				Datagram_size (bits 10-8)
2	Datagram_size (bits 7-0)								
3	Datagram_tag (high order octet)								
4	Datagram_tag (low order octet)								
5	Datagram_offset								

- Datagram\_size: This 11-bit field encodes the size of the entire NSDU before fragmentation plus the size of the basic, contract-enabled or full header. The value of Datagram\_size shall be the same for all fragments of the NSDU.
- Datagram\_tag: This provides the identification for the reassembling device. The sender shall increment Datagram\_tag for successive, fragmented NPDUs. The incremented value of Datagram\_tag shall wrap from 65535 back to zero. This field is 16 bits long, and its initial value shall be random. The value of Datagram\_tag shall be the same for all fragments of an NSDU.
- Datagram\_offset: This field shall be present only in the second and subsequent fragments and shall specify the offset, in units of 8 octets, of the fragment from the beginning of the NPDU payload. (The first fragment of the NSDU has an offset of zero; the implicit value of Datagram\_offset in the first fragment is zero.) This field is 8 bits long.

#### 10.5.5.3 Relation to 6LoWPAN (INFORMATIVE)

The fragmentation header formats in this standard are based entirely on the IETF RFC 4944 format, with no changes. As in IETF RFC 4944, the Datagram\_size field is carried in all fragments, and the Datagram\_offset is expressed in 8-octet units. Fragmentation and reassembly can occur at intermediate devices and are not necessarily end-to-end.

## 11 Transport layer

### 11.1 General

The reference model in this standard is composed of five protocol layers. In this model, transport is the fourth layer, immediately subordinate to the application layer. The transport layer (TL) responds to service requests from the application layer at a transport layer service access point (TSAP) and issues service requests to the network layer at a network layer service access point (NDSAP).

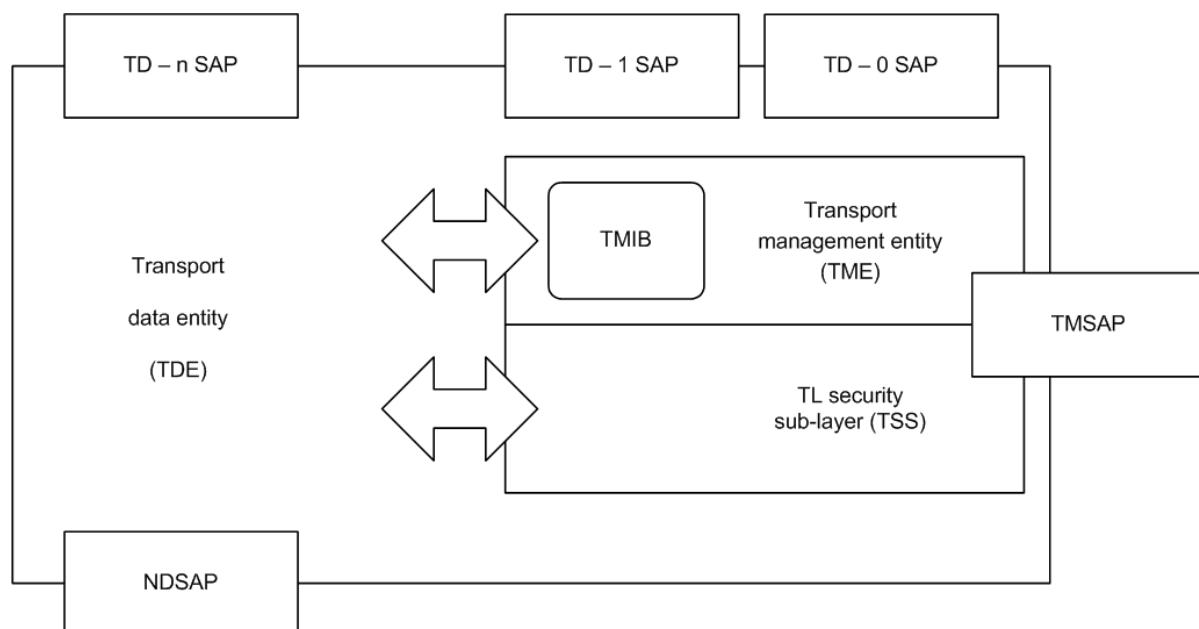
The TL is responsible for end-to-end communication and operates in the communication endpoints (as opposed to the routing devices).

The TL provides connectionless services with optional security:

- The connectionless service extends UDP over IP version 6 (IPv6) as defined by the IETF in IETF RFC 768 and IETF RFC 2460 by providing a better integrity check and additional security for authentication and encryption.
- Security is handled in a similar fashion as in the DL layer, but end-to-end as opposed to hop-by-hop.
- The connectionless service also extends 6LoWPAN as defined by draft-ietf-6lowpan-hc-07. When security is activated, it is possible to compress the UDP header down to one octet by eliding the UDP checksum and relying on the transport message integrity check (TMIC) to guarantee end-to-end integrity.

### 11.2 Transport layer reference model

The TL reference model is shown in Figure 107.



**Figure 107 – Transport layer reference model**

The TL conceptually includes the transport management entity (TME), the transport security sub-layer (TSS), and the transport data entity (TDE). The TME and the TSS share a management service access point, the TMSAP.

The TDE has a service access point (TDSAP) for the device management application process (DMAP), TDE-0; a TDSAP that is reserved for SMAP, TDE-1; and a TDSAP for each user application process (UAP), TDE-2 to TDE-n.

The TME configures and monitors the actions of the transport layer. The Transport Management Information Base (TMIB) (see 11.6.2.5.2) implements abstract information that the TLMO obtains from the TME through the TMSAP. The TSS configures and monitors the

operation of the transport layers security functions. It maintains tables of security structures that are maintained and monitored by the TMSAP. The TDE uses the TSS to perform security operations on protocol data units that pass through the layer.

### **11.3 Transport security sub-layer**

#### **11.3.1 General**

The transport security sub-layer (TSS) is conceptually a service within the TL. TL security can protect the integrity of the application payload which is referred to in this clause as the transport service data unit (TSDU), the transport header, and network addresses, though only the TSDU can be encrypted.

#### **11.3.2 Securing the transport layer**

The TSS within the transport layer reuses many items from the DL security sub-layer. The TSS is responsible for:

- Determining which security level shall be applied to a given secure session based on policies;
- Filtering out non-conforming received TPDU;
- Encrypting/decrypting protected TSDUs; and
- Implementing the determined cryptographic operation for TPDU authentication check.

The functionality of the TL security sub-layer is defined in 7.3.3.

Similar to DL security, TL security supports the default Advanced Encryption Standard (AES) cryptographic block cipher in a generic authenticated encryption block cipher mode called the Counter with CBC-MAC (CCM\*), as defined by IEEE Std 802.15.4-2006 Annex B, which refers to CCM as defined by ANSI X9.63:2001, as well as NIST Pub 800-38C and IETF RFC 3610.

CCM\* enables authentication of a message while encrypting only a part of that message, leaving the rest of the message (typically a header) in the clear. When this feature is enabled for a particular session, the UDP checksum is not used and can be elided in the compressed form of the TPDU as specified in 11.4.3.4, though it shall be present in the expanded form to comply with the UDP standard (IETF RFC 2460).

### **11.4 Transport data entity**

#### **11.4.1 General**

The TDE provides connectionless services based on the User Datagram Protocol (UDP, IETF RFC 768) over IPv6 (IETF RFC 2460) with optional compression as defined by the IETF 6LoWPAN Working Group in draft-ietf-6lowpan-hc-07.

The main steps in TPDU construction are listed below, in order:

- A TSDU is received from the application layer through a TD-SAP. This TSDU may contain a single APDU, or multiple concatenated APDUs.
- The TSDU is protected and timestamped as described in 7.3.3.2.1. The resulting UDP payload comprises a TL security header, the TSDU, and an optional TMIC. The TSDU is encrypted if so indicated in the TL security header. The presence of the TMIC, and its size, is indicated in the TL security header.
- A UDP header is prepended to the UDP payload. The UDP header may be compressed. Compression might involve eliding the UDP checksum within the UDP header, when the UDP payload contains an integrity check. An integrity check within the UDP payload may be a TMIC as indicated in the TL security header, and/or an integrity check embedded in a tunneled payload.

#### 11.4.2 The user datagram protocol for IP version 6 (INFORMATIVE)

The UDP protocol specified in IETF RFC 768 enables a connectionless mode of packet-switched computer communication in the environment of an interconnected set of computer networks.

UDP is essentially transparent to application operation, leaving both control and responsibility for proper network operation to the application layer. Proven, standard-based approaches exist for this purpose. Relevant issues are further documented in 11.4.4.

IETF RFC 768 assumes that Internet Protocol version 4 (IPv4) is used as the underlying protocol and defines the computation of a UDP checksum that covers a UDP “pseudo-header” of information from the IPv4 network header, the UDP header, and the UDP payload. If that computation yields a result of zero, it is changed to 0xFFFF for placement in the UDP header and the value of zero is reserved to indicate that there is no checksum computation.

IETF RFC 2460 specifies the changes to IETF RFC 768 to adapt UDP to IPv6. The changes to UDP are minor and relate to the computation of the UDP checksum that becomes mandatory and includes a new UDP “pseudo-header” that is adapted to IPv6 as shown in Figure 108. Per IETF RFC 2460, IPv6 receivers discard UDP packets containing a zero checksum and log this event as an error.

Source address	Destination address	Upper-layer packet length	zero	Next header
----------------	---------------------	---------------------------	------	-------------

**Figure 108 – UDP “pseudo-header” for IPv6**

In the “pseudo-header”, the value in the field named next header is set to 17 to indicate UDP and the value in the field named upper-layer packet length is the same as the length field in the UDP header and accounts for the size of the whole TPDU starting from the UDP header.

NOTE A “pseudo-header” is also used in security processing, as described in 7.3.3.2.1.

#### 11.4.3 User datagram protocol header transmission and optional compression

##### 11.4.3.1 General

The TL supports uncompressed UDP for both transmission and reception of a TPDU. A device that implements the TL of this standard shall support uncompressed UDP but should compress the UDP header for transmission over the DL network.

Table 221 describes the UDP header encoding.

**Table 221 – UDP header encoding**

octets	Bits (presented in ISA100 convention, which is different from IETF convention)						
	7	6	5	4	3	2	1 – 0
1-2	Source Port: that maps one to one with the source device TDAP						
3-4	Destination: Port that maps one to one with the destination device TDAP						
5-6	Length: the length in octets of this user datagram, including this header and the APDU						
7-8	Checksum: the 16-bit one's complement of the one's complement sum of a UDP “pseudo-header” of information from the IP header, the UDP header, and the APDU, padded with zero octets at the end (if necessary) to make a multiple of two octets. For computing the checksum, the checksum field is first set to zero						

The TL also complies with 6LoWPAN that specifies a method for compressing UDP using a Next Header Compression (LoWPAN\_NHC) format and mandates the use of UDP compression over the DL network. A device that implements the TL of this standard shall thus support all the combinations offered by the LOWPAN\_NHC for compression and expansion of the UDP header.

TSDU capacity depends on a variety of factors, such as the buffering capacity on each side of the session, the TL security level, and network characteristics. The system manager

configures TSDU capacity on a per-contract basis to account for these and any other relevant considerations, through the value of `Assigned_Max_TSDU_Size`, as described in Table 30.

#### **11.4.3.2 Compressing and restoring the user datagram protocol ports**

The compression for UDP ports is described in 3.2. UDP Header Compression, of draft-ietf-6lowpan-hc-07. The port numbers that optimize the compression should be assigned to the busiest applications. The optimum compression is obtained when both the source and destination port numbers start at a base number P and are expressed as P+short\_port, where:

- P is the base port, decimal as 61616, that is, 0xF0B0
- short\_port is a positive integer that is lesser or equal to 15

In that case, the pair of ports can be compressed to one octet that contains the source and destination values of the short\_ports, and the compression mechanism saves 3 octets on the UDP port information.

When it is impossible to maintain both ports within the 0xF0B0 to 0xF0BF range, it is still possible to save one octet in each TPDU if either the source or the destination port is in the 0xF000 to 0xFOFF range, in which case one 0xF0 octet can be elided. It is thus recommended that server applications listen to a port in the 0xF000 to 0xFOFF range. A typical field device will have a DMAP at F0B0 (encoded as 0) and a single UAP at F0B2 (encoded as 2), with F0B1 (encoded as 1) being reserved for the SMAP in the system manager.

#### **11.4.3.3 Eliding and restoring the user datagram protocol length field**

If the UDP header is not compressed, then the NSDU size is equal to the TPDU size that is placed in the UDP header.

If the UDP header is compressed, the UDP Length field is always elided in transmission, and is inferred upon reception from the NSDU size passed by the NL. In that case, the TPDU size is equal to the NSDU Size that is incremented by the difference between the UDP header and the compressed UDP header.

Even if none of the other fields in the UDP header are compressed or elided, compressing the UDP length saves one octet and enables the use of the basic header at the NL.

For example, in the optimal case described in Table 223, the compressed UDP header consumes 2 octets thus saving 6 octets compared with the expanded UDP header. The TDPU size to be placed in the expanded UDP header is thus `NSDU_Size + 6`.

#### **11.4.3.4 Eliding and restoring the user datagram protocol checksum**

The TL complies with the 6LoWPAN rules and operations defined in draft-ietf-6lowpan-hc-07 (4.3.2, Compressing UDP checksum) as follows:

- The authorization to elide the UDP checksum might come from the application layer or the TL security sub-layer. The TL security sub-layer hints by informing whether the TMIC is present or not.
- The source of a packet should elide the UDP checksum in the following cases:
  - Tunneling: The source of the packet is tunneling another PDU that possesses its own integrity mechanism.
  - Transport message integrity check: The source applies the end-to-end TMIC.
  - If the DL network termination point is a backbone router, then the router shall recompute the elided checksum based on the received packet as specified in IETF RFC 2460 and shall place the result of that computation in the reformed UDP header.
  - If the DL network termination point is the destination of the packet and is aware of the presence of the TMIC in the TPDU, then it may omit the UDP checksum operation completely (neither recompute nor check the checksum).
  - A backbone router that forwards an IPv6 packet into the DL network uses the security control field in TL security header to determine whether a TMIC is present or not. When

performing the UDP compression, the backbone router should elide the UDP checksum if the TMIC is present but it shall not elide the UDP checksum if the TMIC is not present. Conversely, the fact that the checksum is not compressed is not an indication that there is no TMIC in the TPDU.

- To enable the UDP checksum compression, 6LoWPAN requires that the TMIC be stronger than the UDP checksum and ensure the end-to-end protection of all the fields that the UDP checksum would protect. For that purpose, this standard defines a TSS “pseudo-header” that is included in the TMIC computation, as described in 7.3.3.2.1. Compared to the UDP “pseudo-header”, the TSS “pseudo-header” includes the UDP ports but does not include the payload length that is protected implicitly by the TMIC. The structure of the TSS “pseudo-header” is defined in Table 47.

In the case of an outgoing TPDU, the source and destination 128-bit addresses are obtained from the contract information; the contract ID itself is passed by the application layer as a parameter associated with the TSDU. The next header is set to 17, meaning UDP. The source port is obtained from the transport layer states associated with the TSAP, whereas the destination port is also a parameter associated with the TSDU. The TSS “pseudo-header” is modeled as being constructed by the TDE and passed to the TL security sub-layer with the TSDU for security processing.

The TL security sub-layer includes the TSS “pseudo-header” in the TMIC computation by prepending it to the TSDU, but the TSS “pseudo-header” is not encrypted should encryption be performed on the TSDU. Once the cryptographic process is completed, the TL security sub-layer shall remove the TSS “pseudo-header” from the processed TSDU, add its own headers and trailers, and return the protected UDP payload to the TDE. The TDE shall complete the TPDU by prepending and optionally compressing the UDP header to form the NSDU that is passed to the NL.

In the case of an incoming TPDU, the source and destination 128-bit addresses and the priority bits are passed by the network layer as parameters associated with the NSDU, and the ports are obtained from the UDP header in the TPDU once the UDP header is expanded. The TDE shall fill the “pseudo-header”, remove the UDP header from the NSDU, and pass the resulting protected UDP payload together with the “pseudo-header” and the priority bits to the TL security sub-layer for security processing.

Since an NPDU might be received out of sequence in a fashion that depends on its priority settings (see Table 205), the TL security sub-layer needs the priority information in order to reduce the size of the lookback tables and optimize the capability to validate the TPDU within limited memory resources. The priority information is associated with an NSDU in the N-DATA.indication received by the transport layer from the network layer and the TLE forwards that information to the TL security sub-layer with the TPDU.

The TL security sub-layer shall strip its headers and trailers from the protected TSDU, prepend the TSS “pseudo-header”, UDP ports, and security headers (see Figure 41), and then perform the security verifications. If an integrity check is applied, the TL can verify whether any of the critical parts of the network and transport information was modified during transport.

**NOTE 1** If the UDP upper layer checksum field is elided, it is up to the receiving endpoint on the DL subnet to recompute the UDP checksum as part of the 6LoWPAN expansion, if necessary. This is required in particular for a middlebox, such as a backbone router, that forwards the uncompressed NPDU, but useless in a transport endpoint.

**NOTE 2** If the UDP checksum is not elided, then the UDP checksum shall be computed in transmission after the security operation is complete and the security fields are filled. The UDP payload that is used for the UDP checksum shall include all data from the UDP header to the end of the TPDU, including the security fields, application data, and MIC fields, whether the application data is encrypted or not. In reception, it shall be verified by the TDE prior to TL security sub-layer operation unless it is known by the TDE that the TL security sub-layer verifies the MIC, in which case the checksum may be ignored.

#### 11.4.4 Transport layer data service access points and user datagram protocol ports

A device autoconfigures one 128-bit IPv6 link local address based upon its EUI-64 address. After the join process is complete, a device also owns at least one 128-bit IPv6 global address. This standard allows, but does not require, a device to support more than one IPv6

global address. For a given IPv6 address, a UDP port shall be associated with no more than one user application process (UAP).

The port is used as local port for all transmissions from/to that UAP using that IPv6 address. Further multiplexing between UAP objects shall be handled by the application within its payload and is transparent to the transport layer. As a result, only a limited number of ports are actually used in the system, and there is no dynamic port allocation after the (re-)initialization phase of the applications.

Each UAP shall be responsible for knowing its UDP port number and claiming it from the transport layer. At that time, a TLSAP shall be mapped on a one-to-one basis with the UAP and the local port for the given IPv6 address. An application process address shall consist of a device address concatenated with its TLSAP identifier.

#### **11.4.5 Good network citizenship (INFORMATIVE)**

(n)-SDUs are exchanged with the upper and lower layers through (n)-DATA primitives. An implementation can report transient congestion in a lower layer via a specific completion code in (n)-DATA.conf all the way up the protocol suite. Upon receiving the completion code, the application layer should either delay by an exponential backoff amount or simply drop the (n)-SDU.

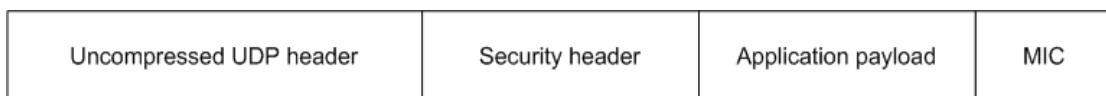
Since the transport layer is based on UDP, it is desirable that the UAPs support TCP-Friendly Flow Control or TCP-Friendly Rate Control (TFRC, IETF RFC 3448), depending on the nature of the flow, in order to protect future uses of the network and share the backbone fairly.

This is typically achieved by implementing a blocking protocol at the application layer that enforces a round-trip time (RTT) between PDUs or uses RTT to compute its loss recovery timer. In the case of periodic samples, this is achieved by reserving bandwidth or by keeping the sampling period above the initial RTT value defined for TCP of 3 s (see IETF RFC 2988), though higher sample rates may be used in some applications.

### **11.5 Transport layer protocol data unit encoding**

#### **11.5.1 General**

A TPDU exchanged between the transport layers of the two endpoints of a communication shall be composed of a UDP header in its uncompressed form, a security information header, the TSDU, and a TMIC, as shown in Figure 109.



**Figure 109 – Transport layer protocol data unit**

The NSDU passed by the transport to the network layer over the NDSAP is the result of the optional UDP header compression applied to the TPDU. If compression was applied, the 6LoWPAN\_NHC for UDP encoding is the first octet of the NSDU.

The pair of IPv6 addresses, the transport type (UDP=17) and the size of the NSDU are not present in the NSDU and are passed as parameters. Whether 6LoWPAN\_NHC compression took place is also passed as a parameter.

#### **11.5.2 Header compression – User datagram protocol encoding**

When the UDP header is not compressed, the NSDU is identical to the TPDU and the UDP header shall be present in full without a 6LoWPAN\_NHC for UDP encoding octet.

If any of the UDP header fields is compressed, a 6LoWPAN\_NHC for UDP encoding octet shall be built to describe the compression. The NSDU shall be formed by replacing the UDP header at the beginning of the TPDU with the encoding octet followed by the compressed data. The LoWPAN\_NHC is structured as in Table 222.

**Table 222 – UDP LowPAN\_NHC encoding**

octets	bits (presented in ISA100 convention, which is different from IETF convention)							
	7	6	5	4	3	2	1	0
1	1	1	1	1	0	C	P	

The C (checksum compression) and P (ports compression) fields are defined as follows:

- C field (1 bit):
  - 0: All 16 bits of the UDP checksum are carried in-line.
  - 1: All 16 bits of the UDP checksum are elided.
- P field (2 bits):
  - 00: All 16 bits for both source port and destination port are carried in-line.
  - 01: All 16 bits for source port are carried in-line. The first 8 bits for the destination port are 0xF0 and are elided. The remaining 8 bits of the destination port are carried in-line.
  - 10: The first 8 bits of the source port are 0xF0 and are elided. The remaining 8 bits of the source port are carried in-line. All 16 bits for the destination port are carried in-line.
  - 11: The first 12 bits of both source port and destination port are 0xF0B and elided. The remaining least significant 4 bits for each are carried in-line.

Fields carried in-line (in part or in whole) appear in the same order as they do in the UDP header format as specified in IETF RFC 768.

When the highest degree of compression is achieved, only the compressed short\_port numbers are carried after the 6LoWPAN\_NHC for the UDP encoding octet as shown in Table 223.

**Table 223 – Optimal UDP header encoding**

octets	bits (presented in ISA100 convention, which is different from IETF convention)							
	7	6	5	4	3	2	1 – 0	
1	1	1	1	1	0	1 (checksum is elided)	1 – 1 (source and destination ports are compressed)	
2	source short_port					destination short_port		

The expanded values for the compressed source port and destination ports shall be calculated using the formula:

$$\text{UDP\_port\_number} = P + \text{short\_port}$$

where short\_port is the 4-bit partially compressed port number and P is the number 61616 (0xF0B0). In other words, short\_port forms the 4 least significant bits of the UDP port number.

At the time of the join process messages, the field device does not have the relevant crypto material and will not be able to compute a MIC. As the MIC is omitted, the device shall compute the UDP checksum as prescribed by IETF RFC 768 and IETF RFC 2460 and shall use the link local IPv6 addresses to compute the checksum and transmit; if the UDP ports can be compressed, the encoded UDP header is formatted as represented in Table 224.

**Table 224 – UDP header encoding with checksum**

octets	bits										
	7	6	5	4	3	2	1 – 0				
1	1	1	1	1	0	0 (checksum is inline)	1 – 1 (source and destination ports are compressed)				
2	source short_port			destination short_port							
3											
4	UDP Checksum										

### 11.5.3 Transport layer security header

For information on the TL security header, encoding, and decoding, see 7.3.3.6.

## 11.6 Transport layer model

### 11.6.1 General

The transport layer provides two interfaces:

- A transport layer data entity (TDE) service access point (TDSAP), one for each UAP and one for the DMAP.
- A transport layer management entity (TME) service access point (TMSAP) for the device manager (DMAP).

These interfaces are illustrated in the protocol reference model of this standard, shown in Figure 16.

Upper layers use the TDSAP to exchange data communicated through the protocol suite via the transport layer. There shall be one instance of the TDSAP for each UAP, plus an instance for the DMAP.

The TDSAP shall include a multiplexer function that adapts the address namespace of the application layer to the native address namespace of the transport layer. The DMAP shall use the TMSAP to configure the transport layer and monitor its operation.

The transport layer communicates with the network layer using the NDSAP.

### 11.6.2 Data services

#### 11.6.2.1 General

For illustration purposes, an example set of primitives is provided herein. The transport layer offers an unconnected service based on the UDP model.

The transport layer uses the interface supplied by the TDSAP to transmit and receive protocol data units with the ASL.

The TDSAP transfers the PDU, along with control and status information parameters.

#### 11.6.2.2 T-DATA.request

##### 11.6.2.2.1 General

T-DATA.request instructs the transport layer to transmit a protocol data unit.

##### 11.6.2.2.2 Semantics of the service primitive

The semantics of T-DATA.request are as follows:

```
T-DATA.request
(
  ContractId
  TSDU_size
  TSDU
  Priority
  DE
  TDSAP
  Destination_Port)
```

Table 225 specifies the elements for T-DATA.request.

**Table 225 – T-DATA.request elements**

Element name	Element identifier	Element scalar type
ContractId (identifies the contracted transport layer resources associated with the protocol data unit)	1	Type: Unsigned16 Valid value set: [0, $2^{16}$ -1]; 0 indicates no contract
TSDU_size (size in octets of the protocol data unit passed from the ASL)	2	Unsigned16
TSDU (the TSDU to be transmitted)	3	Type: Data
Priority (identifies per APDU priority of this PDU)	4	Type: Unsigned2
DE (identifies the Discard Eligibility of this PDU)	5	Type: Unsigned1
TDSAP (ID of the TDSAP that grants UDP service over SourcePort)	6	Type: Unsigned16
Destination_Port (UDP destination (remote) port in the outgoing TPDU)	7	Type: Unsigned16

The TL maintains a table indexed by the TDSAP that contains (implicitly or explicitly) the local address and (explicitly) the local port for transmission. That table is accessed to obtain the source address and port for transmission over a given TDSAP. The destination address is obtained from the Destination\_Address field in the contract information, indexed by the contract ID.

There is no state in the transport layer related to the remote port; thus, information is passed by the user application process and placed in the TPDU without checking.

The message priority transmission option communicates the transmission level priority that is used by the network. The priority settings are not used in the transport layer, but they are passed on through the network layer to the DL, where they are used to select the priority class in the DL header.

#### 11.6.2.2.3 Appropriate usage

The ASL invokes the T-DATA.request primitive to transmit an APDU as a TSDU.

#### 11.6.2.2.4 Effect on receipt

On receipt of the T-DATA.request primitive, the transport layer looks up the security level in the KeyDescriptor to determine the security processing required, constructs the transport layer header, forms the transport layer protocol data unit, and generates the N-DATA.request to the network layer SAP.

#### 11.6.2.3 T-DATA.confirm

##### 11.6.2.3.1 General

T-DATA.confirm is used by the transport layer to respond to a T-DATA.request. The confirmation is immediate and tells the ASL that the request was successful or resulted in an error. Error conditions include items such as an unrecognized contract ID, a PDU length that is wrong or too big, or an internal transient error such as network congestion.

##### 11.6.2.3.2 Semantics of the service primitive

The semantics of T-DATA.confirm are as follows:

```

T-DATA.confirm      (
    ContractId
    TDSAP
    status
)

```

Table 226 specifies the elements for T-DATA.confirm.

**Table 226 – T-DATA.confirm elements**

Element name	Element identifier	Element scalar type
ContractId (identifies the contracted transport layer resources associated with the protocol data unit)	1	Type: Unsigned16 Valid value set: [0, $2^{16}-1$ ]; 0 indicates no contract
TDSAP (ID of the TDSAP that grants UDP service over SourcePort)	2	Type: Unsigned16
Status (the result of the T-DATA.request primitive)	3	Type: Enumeration Valid value set: (see Table 227)

Table 227 provides the status codes for T-DATA.confirm.

**Table 227 – T-DATA.confirm status codes**

Name	Description
SUCCESS	PDU accepted
TRANSIENT_FAILURE	PDU rejected, but can be retried after a short period of time
FAILURE	Generic failure: PDU rejected without explicit reason.
INVALID_CONTRACT	Specific failure: Unrecognized contract ID; PDU rejected.
INVALID_LENGTH	Specific failure: PDU passed is longer than the <b>Assigned_Max_TSDU_Size</b> ; PDU rejected.
PORT_ERROR	Specific failure: SourcePort is not registered for TDSAP; PDU rejected.
SAP_ERROR	Specific failure: Unknown TDSAP; PDU rejected.

#### 11.6.2.3.3 When generated

The transport layer generates T-DATA.confirm in response to a T-DATA.request. The T-DATA.confirm returns synchronously either a status of success or the appropriate error code.

#### 11.6.2.3.4 Appropriate usage

On receipt of the T-DATA.confirm primitive, the ASL is notified of the result of its request to transmit an APDU.

#### 11.6.2.4 T-DATA.indication

##### 11.6.2.4.1 General

T-DATA.indication is used to transfer a protocol data unit to the ASL. It is generated when a protocol data unit has been successfully received from the network layer and processed by the transport layer.

A protocol data unit that fails the security processing or has an unknown TSAP ID is not passed to the ASL. Such errors are logged in the transport layer PIB and are not reported by the T-DATA service.

##### 11.6.2.4.2 Semantics of the service primitive

The semantics of T-DATA.indication are as follows:

```
T-DATA.indication ( SrcAddr
                    SrcPort
                    TSDU_size
                    TSDU
                    ECN
                    TDSAP
                    transportTime
)
```

Table 228 specifies the elements for T-DATA.indication.

**Table 228 – T-DATA.indication elements**

Element name	Element identifier	Element scalar type
SourceNetworkAddress (IP address of the remote end)	1	Type: 128-bit address
SourcePort (UDP source port in incoming TPDU)	2	Type: Unsigned16
TSDU_size (size in octets of the TSDU passed from the ASL)	3	Type: Unsigned16
TSDU (the APDU to be received)	4	Type: Data
ECN: Explicit Congestion Notification bits	5	Type: Unsigned2
TDSAP (ID of the TDSAP that grants UDP service and matches a local Port)	6	Type: Unsigned8
Transport Time (Expressed in seconds, time of flight as seen from the transport security layer)	7	Type: Unsigned16 (see pduMaxAge)

The TL maintains a table that contains the TDSAP, which is indexed by (implicitly or explicitly) the local address and (explicitly) the local port for reception. That table is accessed to find the TDSAP used to pass the TSDU to the user application process.

#### 11.6.2.4.3 Appropriate usage

The transport layer invokes T-DATA.indication to notify the application support sub-layer of a received TSDU.

#### 11.6.2.4.4 Effect on receipt

On receipt of T-DATA.indication, the application support sub-layer processes the received TSDU.

#### 11.6.2.5 Management services

##### 11.6.2.5.1 General

The transport layer management service is controlled by the transport layer management object (TLMO) in the DMAP. The TLMO controls the transport layer functionalities by:

- Measuring transport layer latency and making the related adaptations to comply with latency requirements dynamically.
- Collecting operational parameters.

The TLMO uses the interface supplied by the TMSAP to configure and control the operation of the transport layer. The TME inside the TL implements a TMIB to maintain the states that are necessary to implement the TMSAP.

##### 11.6.2.5.2 Attributes

Table 229 specifies the attributes of the TLMO.

**Table 229 – TLMO attributes**

<b>Standard object type name: Transport layer management object</b>				
<b>Standard object type identifier: 122</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Reserved	1	Reserved by the standard		
MaxNbOfPorts	2	Number of active ports	Type: Unsigned8	The minimum value covers a typical device with a single DMAP and a single UAP TDSAPs
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: Device-dependent	
			Valid value set: 2 – 255	
TPDUin	3	Counter reporting the number of TPDUs received.	Type: Unsigned32	Incremented with each TPDU received from the remote transport layer
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set: 0– 0xFFFFFFFF	
TPDUinRejected	4	Counter reporting the number of rejected TPDUs	Type: Unsigned32	Incremented with each data unit received from the remote transport layer but not accepted (e.g., for security reasons). Note: there is no such counter at DSMO
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set: 0– 0xFFFFFFFF	
TSDUout	5	Counter reporting the number of TSDU passed to the application layer	Type: Unsigned32	Incremented with each TPDU received from the remote transport layer from which a TSDU was passed to the application layer
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set: 0– 0xFFFFFFFF	
TSDUin	6	Counter reporting the number of TSDUs received	Type: Unsigned32	Incremented with each service data unit received from the application layer
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set: 0– 0xFFFFFFFF	
TSDUinRejected	7	Counter reporting the number of rejected TSDUs	Type: Unsigned32	Incremented with each service data unit received from the application layer that failed the checks and could not be passed down to the network layer
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set: 0– 0xFFFFFFFF	
TPDUout	8	Counter reporting the number of TPDU passed to	Type: Unsigned32	Incremented with each TSDU
			Classification:	

<b>Standard object type name: Transport layer management object</b>				
<b>Standard object type identifier: 122</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
		the network layer	Dynamic Accessibility: Read only Initial default value: 0 Valid value set: 0–0xFFFFFFFF	received from the application layer and forwarded to the remote transport layer
IllegalUseOfPortAlertDescriptor	9	Used to change the priority of IllegalUseOfPort alert; this alert can also be turned on or turned off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Initial default value: Alert report disabled = True Alert report priority = 8 (medium) Valid value set: See type definition	
TPDUonUnregisteredPort AlertDescriptor	10	Used to change the priority of TPDUonUnresiteredPort alert; this alert can also be turned on or turned off	Type: Alert report descriptor Classification: Static Accessibility: Read/write Initial default value: Alert report disabled = True Alert report priority = 4 (low) Valid value set: See type definition	

For each attribute, the TLMO provides read and write methods available to the DMAP. Those methods are implemented by requesting TME services across the TMSAP.

#### 11.6.2.5.3 Methods

In addition to the read and write service for the attributes, additional TLMO methods provide access to TME services across the TMSAP.

Table 230 describes the reset method.

**Table 230 – Transport layer management object methods – Reset**

<b>Standard object type name: Transport layer management object</b>			
<b>Standard object type identifier: 122</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method Description</b>	
Reset	1	Reset the transport to initial states	
	<b>Input Argument</b>		
	Argument #	Argument name	Argument type (data type and length)
	1	Forced	Boolean
	Forced means that all data are reformatted without any flow with other entities. SAP related tables are emptied, related memory is freed and all counters are set to 0		
	<b>Output Arguments</b>		
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Status	Unsigned8	0 = success

Table 231 describes the halt method.

**Table 231 – Transport layer management object methods – Halt**

<b>Standard object type name: Transport layer management object</b>			
<b>Standard object type identifier: 122</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method Description</b>	
Halt	2	Halts a port and places it back in initial state. Like a reset, but scoped to one UDP port only. All traffic is interrupted on that port and the SAP needs to be reopened for that port to become operational again. The SAP related table entries are emptied and related memory is freed	
	<b>Input Arguments</b>		
	Argument #	Argument name	Argument type (data type and length)
	1	DeviceAddress	128-bit address
	2	PortNumber	Unsigned16
	<b>Output Arguments</b>		
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Status	Unsigned 8	0 = success, 1 = generic failure 2 = bad port number

Table 232 describes the PortRangeInfo method.

**Table 232 – Transport layer management object methods – PortRangeInfo**

<b>Standard object type name: Transport layer management object</b>			
<b>Standard object type identifier: 122</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method Description</b>	
PortRangeInfo	3	Reports the UDP ports range information	
	<b>Input Arguments</b>		
	Argument #	Argument name	Argument type (data type and length)
	1	DeviceAddress	128-bit address
	<b>Output Arguments</b>		
	Argument #	Argument name	Argument type (data type and length)
	1	Status	Unsigned 8
	2	NbActivePorts	Unsigned 16
	3	FirstActivePort	Unsigned 16
	4	LastActivePort	Unsigned 16

Table 233 describes the GetPortInfo method.

**Table 233 – Transport layer management object methods – GetPortInfo**

<b>Standard object type name: Transport layer management object</b>			
<b>Standard object type identifier: 122</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method Description</b>	
GetPortInfo	4	Reports the UDP port information for a given UDP port or the first active UDP port	
	<b>Input Arguments</b>		
	Argument #	Argument name	Argument type (data type and length)
	1	DeviceAddress	128-bit address
	2	PortNumber	Unsigned16
	<b>Output Arguments</b>		
	Argument #	Argument name	Argument type (data type and length)
	1	Status	Unsigned8
	2	PortNumber	Unsigned16
	3	UID	Unsigned32
	4	Compressed	Boolean
	5	TPDUsInOK	Unsigned32
	6	TPDUsInKO	Unsigned32
	7	TPDUsOutOK	Unsigned32
	5	TPDUsOutKO	Unsigned32

Table 234 describes the GetNextPortInfo method.

**Table 234 – Transport layer management object methods – GetNextPortInfo**

<b>Standard object type name: Transport layer management object</b>			
<b>Standard object type identifier: 122</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method Description</b>	
GetNextPortInfo	5	Reports the UDP port information for a given UDP port or the first active UDP port	
<b>Input Arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	DeviceAddress	128-bit address	This device address
2	PortNumber	Unsigned16	The previous port from which info is requested
<b>Output Arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	Status	Unsigned8	0 = success, 1 = failure
2	PortNumber	Unsigned16	The port for which info is reported
3	UID	Unsigned32	Owner application ID
4	Compressed	Boolean	Whether compression applies to this port
5	TPDUsInOK	Unsigned32	Number of TPDUs accepted over the port
6	TPDUsInKO	Unsigned32	Number of TPDUs rejected over the port
7	TPDUsOutOK	Unsigned32	Number of TPDUs sent over the port
5	TPDUsOutKO	Unsigned32	Number of TPDU transmission failures

#### 11.6.2.5.4 Alerts

Table 235 describes the alert to indicate illegal use of a port by an application.

**Table 235 – Transport layer management object alert types – Illegal use of port**

<b>Standard object type name(s): Transport layer management object</b>					
<b>Standard object type identifier: 122</b>					
<b>Defining organization: ISA</b>					
<b>Description of the Alert: Alert to indicate illegal use of a port by an application</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority (Enumerated: high, med, low, journal only)	Value data type	Description of value included with alert
0 = Event	1= Communication diagnostic	0 =IllegalUseOfPort	8 = Medium	Type: Unsigned16	16-bit port number
				Initial default value: N/A	
				Valid value set: 0-0xFFFF	

Table 236 describes the alert to notify of a received TSDU that addresses an unregistered port.

**Table 236 – Transport layer management object alert types – TPDU received on unregistered port**

<b>Standard object type name(s): Transport layer management object</b>					
<b>Standard object type identifier:</b> 122					
<b>Defining organization:</b> ISA					
<b>Description of the Alert:</b> Alert to notify of a TPDU received on unregistered port					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority (Enumerated: high, med, low, journal only)	Value data type	Description of value included with alert
0 = Event	1 = Communication diagnostic	1=TPDUonUnregisteredPort	4 = Low	Type: OctetString	First 40 octets of the TPDU
				Initial default value: N/A	
				Valid value set: N/A	

Table 237 describes the alert to notify of a received TPDU that does not match security policies.

**Table 237 – Transport layer management object alert types – TPDU does not match security policies**

<b>Standard object type name(s): Transport layer management object</b>					
<b>Standard object type identifier:</b>					
<b>Defining organization:</b> ISA					
<b>Description of the Alert:</b> Alert to notify of a TPDU that does not match security policies					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority (Enumerated: high, med, low, journal only)	Value data type	Description of value included with alert
0 = Event	1 = Communication diagnostic	2=TPDUoutOfSecurityPolicies	2 = Journal	Type: OctetString	First 40 octets of the TPDU
				Initial default value: N/A	
				Valid value set: N/A	

## 12 Application layer

### 12.1 General

The application layer (AL) defines software objects to model real-world objects, and also defines the communication services necessary to enable object-to-object communication between distributed applications in an open, interoperable application environment compliant with this standard. This standard does not define the operation of the distributed applications themselves; that is, neither the local operation of the application itself, such as the manner in which an application acquires the values of the object attributes it supports for access, nor direction regarding how and when an application applies the models and/or the services defined herein, are addressed by this standard.

NOTE 1 For example, a real-world analog input is modeled as an AnalogInput object. The AnalogInput object may communicate its process variable to a corresponding interested party by using the AL-provided publish service.

The AL supports wireless devices in the field, as well as gateways that integrate a wireless network compliant with this standard and its devices with a host control system.<sup>5</sup>

The application model in this standard is specifically designed to satisfy the constraints of wireless communication environments.

NOTE 2 An object oriented AL approach is used for the following key reasons:

- (1) A command based protocol can be designed to conform to the object model defined by this standard by describing the commands as separate object methods. That is, a command based application can be modeled using the object model in this standard.
- (2) The object model supports well accepted architectural principles of logical information separation. For example, management information is logically separate from operating data. Operational information for independent variables is logically separate. In order to maintain separation of information, the protocol is required to identify the corresponding object. This adds a one octet overhead to identify the object, which was deemed to be a more than reasonable approach for architectural separation of information.

### 12.2 Energy considerations

The need to extend battery life makes energy-efficient messaging extremely important. The use of battery power or energy scavenging/harvesting techniques for connected field devices requires additional considerations in communication layer design, compared to the approach taken for wired devices. Not only does every communicating layer need to consider device resource availability, but also it needs to consider energy consumption minimization (within architecturally appropriate constraints of course) in order to extend battery life or to operate within the scavenging/harvesting budget.

Since energy is consumed by message processing, as well as by the fundamental control operation of the device, it is necessary to balance communications efficiency with employing proven and well-accepted architectural principles of information separation as well as message processing efficiency. The native application model in this standard is defined to meet these needs.

### 12.3 Legacy control system considerations

Wireless networks compliant with this standard may be connected to legacy control systems.

NOTE 1 A model that integrates with existing systems permits the reuse of existing and proven tools and interfaces, and also reduces overall development and test time, resulting in earlier production of robust implementations.

An application process in a native device communicates over the network using only ASL-defined services and payloads. An application process in a non-native device requires communication of constructs that are not defined by the ASL service payloads. This communication from the non-native device over the network defined by this standard is accomplished by using the subset of standard services defined in this standard that are

---

<sup>5</sup> See 5.2.6.2 for a more complete discussion of the roles supported by this standard.

defined to support communication of payloads not explicitly and entirely defined by this standard.

The native AL defines special objects and services to support non-native protocol tunneling to meet system integration needs.

The set of defined application objects that support non-native defined payloads are the tunnel object and the interface object. The set of standard-defined application services that support non-ISA defined payloads are the tunnel service, which is used for aperiodic communication, and the publish service using non-native mode, which is used for periodic communications. For example, the payload of the tunnel service for aperiodic messaging in order to encapsulate data constructed by a legacy system is not defined within WISN.

NOTE 2 One way to achieve a synthesis with existing systems is to create an energy- and resource-optimized version of a wire-oriented existing legacy approach by mapping the legacy model to the model in this standard and directly employing the native AL. Such mappings may be defined by individual protocol consortia, such as the HART Communication Foundation (HCF), Fieldbus Foundation H1 (FF-H1), PROFIBUS Nutzerorganisation (PNO), ODVA, which supports managed protocols such as CIP (Common Industrial Protocol), and others.<sup>6</sup> Another way to achieve synthesis is to use a protocol tunnel through the native AL. For both approaches, energy and resource implications should be considered.

NOTE 3 Whichever method is chosen, the higher-level system needs to be able to communicate with wireless devices within a network compliant with this standard.

NOTE 4 Electronic device description files can be used to meet this need. For this standard, device description language (DDL) or extended device description language (EDDL) may be used to describe native devices.

NOTE 5 See Annex R for further details regarding host system interface.

## 12.4 Introduction to object-oriented modeling

### 12.4.1 General

An object model is a protocol-, platform-, and language-neutral means of describing and distinguishing components (system elements) that have a unique identity. Objects separate the world into meaningful and manageable pieces. Not only do object definitions promote modularity, but they also promote component reusability. An object can represent anything that has state and behavior; objects expose attributes to represent state, and provide methods that operate on the object's state to effect particular behaviors. For example, device-specific methods that may be supported by a DMAP may include device-specific self-test methods or device reset methods.

### 12.4.2 Object-to-object communication concept

From the user's point of view, AL communication occurs from one object in an application process to another object in an application process. Concepts of polymorphism allow the same communication techniques to be applied to this standard's industry-independent user application objects as to this standard's industry-dependent objects, as well as to this standard's management objects.

In keeping with this principle, the application model defines both an object model and a communication interaction model (service and protocol). The application model also supports multiple application processes within a device, each of which may contain multiple standard objects. This enables this standard to meet specific market needs, such as for process industries or factory automation, as well as to enable support for both single processor and multi-processor device architectures.

The application object may, for example use the services of the AL to:

- Read the value of an attribute of a remote object;

---

<sup>6</sup> HART, FF-H1, PROFIBUS, and ODVA are the trademarks of various trade organizations. This information is given for the convenience of users of the standard and does not constitute an endorsement of the trademark holders or any of their products. Compliance to this profile does not require use of the registered trademark. Use of the trademarks requires permission of the trade name holder.

- Write the value of an attribute of a remote object;
- Request execution of an object-specific method of a remote object;
- Report an alert related to a remote object;
- Acknowledge an alert reported by a remote object;
- Publish data to a remote object by using scheduled communication bandwidth;
- Tunnel a non-WISN-native application message to a remote object.

Coding for these services can be found in 12.23.1.4.

#### **12.4.3 Application layer structure**

The AL is divided into the two sub-layers, the upper application layer (UAL) and the application sub-layer (ASL), as shown in Figure 16. There is a one-to-one relationship between an ASLDE-SAP and a TLDE-SAP.

The UAL contains the application processes for the device. These processes may be represented as a user application process (UAP) or as a management process (MP), such as the DMAP or other logical management application such as a security management application.

UAPs may be used, for example, to:

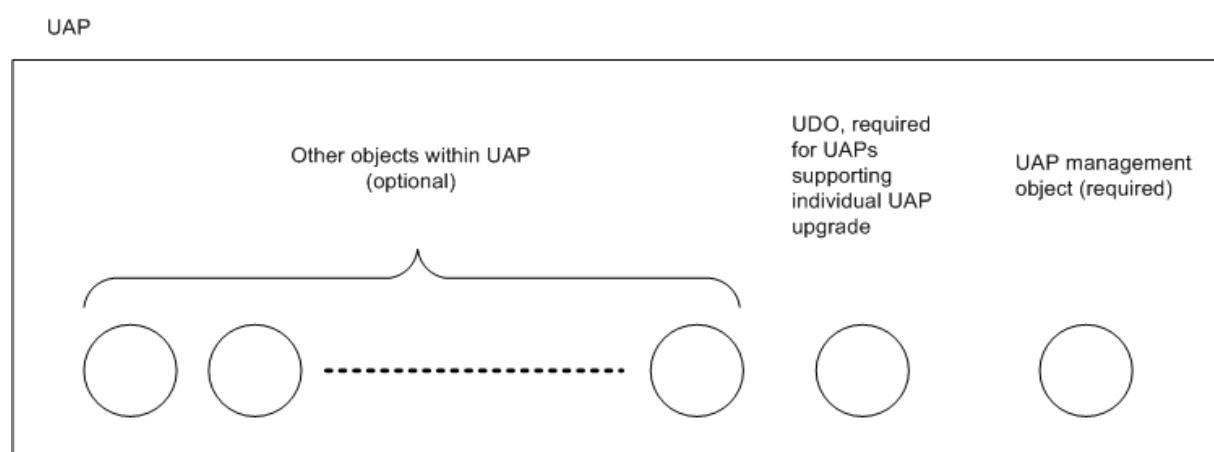
- Handle input and/or output hardware;
- Distribute communications to a set of co-resident UAPs within a device (proxy function);
- Support tunneling of a non-native (e.g., control system legacy) protocol compatible with the network environment of this standard; and/or
- Perform a computational function.

A UAP may perform an individual function or any combination of functions. How a UAP accomplishes these functions internally is beyond the scope of this standard. The AL is concerned with application-specific message content, the externally visible behavior of the standard objects contained within the UAP, and the logical interfaces to the ASL that represent UAP communication to and from the lower communication protocol suite of this standard. UAL processes may contain one or more objects that communicate with one another over the network described in this standard, using the standard services provided by the ASL.

NOTE How the ASL implements internal message routing or inter-application communication within a device (within a UAP, across UAPs in a common processor, across UAPs in different physical processors of the same device, or between a UAP and an MP) is a local matter and hence is outside the scope of this standard.

#### **12.4.4 User application process structure**

Figure 110 depicts the overall structure of a UAP as defined by this standard.



**Figure 110 – User application objects in a user application process**

Representation of the applications and their functions by standard object definitions allows uniform management and construction of distributed applications. The UAP management object identifies the UAP to the standard-compliant network and allows visibility of and/or control over certain operational aspects of the UAP as a whole. This UAP management object has a reserved object identifier of 1 (one). If a UAP supports individual upgrade, the UAP shall also contain a standard UploadDownload object (UDO) to support UAP upgrade. This UDO has a reserved object identifier of 2 (two). If a UAP does not support individual UAP upgrade, the UAP shall not contain an object instance with object identifier of 2 (two). Additional objects also may be contained within a UAP in order to provide application-specific functionality to the UAP.

NOTE 1 Only the UAPMO is required in order to represent the UAP to the communication system.

NOTE 2 Other objects may, for example, be statically or dynamically instantiated within the UAP.

NOTE 3 It is outside the scope of this standard to define what happens to the data contained within a UAP when the UAP is upgraded. For example, the data may be reinitialized, or it may be preserved as it is, or it may be preserved and manual data upgrade may be required, or it may be automatically upgraded.

The interaction model describes inter-object communication, including message classification and messaging formats. The ASL contains services that support object-oriented communication and routing to the appropriate destination object within a UAP, across the network. This interaction maps the ASL to the services provided by the lower communication protocol suite layer (see Table 281 and Table 282 indicating AL use of transport layer (TL) services and qualities of service). Between the UAL and the ASL is an ASL data entity service access point (ASLDE-n SAP).

ASL-specific management is also locally supported via a separate management SAP.

## 12.5 Object model

The AL defined by this standard takes advantage of object-oriented modeling concepts to support both native protocol and non-native (legacy) protocol tunneling within applications. Non-native protocol tunneling is achieved by a specialized user application process (UAP) that includes one or more tunnel objects (TUN) and protocol translation facilities. This UAP consists of exactly one UAP management object to enable uniform system and network management of the UAP, plus one or more TUNs to send/receive encapsulated messages being tunneled to the UAP. Other native objects defined by this standard may also be used within a tunneling UAP. For example, tunneling may be used for wireless device-to-wireless device communication, as well as for wireless device-to-gateway communication.

NOTE 1 Addressing constructs other than these are outside the scope of the standard. Legacy protocol APDU (as opposed to native APDU) constructs are preserved by means of encapsulation when application tunneling as defined by this standard is used.

An object-oriented approach is used to encapsulate data (attributes) and functionality (methods and internal state) for re-use and consistency. Objects are individually addressable using an object identifier that is unique within the application. This unique identifier allows the AL to route messages to the appropriate object destination. Each message is interpreted and acted on by the destination object based on the message context and content. Object operation is described in terms of the network-visible operation of the destination object that is the target of the AL service.

This standard defines standard object identifiers.

NOTE 2 The complement of standard identifiers supported by a device is indicated by the version of this standard that is supported. The supported version is available from the DMAP. See Clause 6 for further details.

An object instance that is accessible within an application process is distinguished from another instance of the same object class in the same application process by its object identifier. Different object instances may also have different attribute values and/or the conditional attributes contained in its set of supported attributes.

As an example, an application process may contain two instances of the AnalogInput object. The object instances within the application process can be distinguished by their object identifier. Each object may support different values for scaling, and also may require a different complement of alarms to be reported. Hence, the instances may have a different complement of analog descriptor attributes supported by each instance.

The objects defined in the UAPs and MPs of this standard adhere to traditional object modeling concepts; specifically, these objects contain attributes, and appropriate object-specific methods, if applicable, are defined. The AL defines standard objects to enable interoperability via access to standard attributes and invocation of standard methods.

Standard objects can be classified into usage profiles to meet the needs of particular industries.

Standard management objects are expected to be always available. Application user objects may be statically instantiated, or they may be dynamically instantiated as the result of a download operation. Standard objects are extensible in the following ways:

- Industry-specific standard object types may be added.
- Vendor-specific object types may be added.
- Industry-specific attributes may be added to standard objects.
- Vendor-specific attributes may be added to standard objects.
- Industry-specific methods may be added to objects via industry-specific profiles.
- Vendor-specific methods may be added to objects.
- Industry-specific profiles of object types may be defined, such as a process industry profile, a factory automation industry profile, and other profiles.

## **12.6 Object attribute model**

### **12.6.1 General**

Objects defined by this standard support two kinds of attributes, object key attributes and named attributes.

In this standard, a resource is represented as an object, and identified by an object key attribute, which is a numeric value.

NOTE 1 Future releases of this standard may support an object key attribute using an alphanumeric representation of the resource. Directory services may then be provided to locate the resource and to resolve the external alphanumeric name form to an internal numeric form.

An attribute indicates an accessible element representing a property or characteristic of a resource. In this standard, an attribute is represented by a unique numerical value that uniquely identifies the attribute relative to the containing object. The supported range of the valid values for an attribute identifier is from 0 to 4095.

This standard defines standard object attributes. Additional standard attributes may be defined in the future.

NOTE 2 The complement of standard attributes supported is established by the version of this standard that is supported. The version of the standard that is supported is available from the DMAP.

Exposing the resource elements as attributes of an object allows the state of the object to be determined and also allows the behavior of the object to be modified. A resource attribute is defined by:

- Its influence on object behavior;
- The set of values it can take (as constrained by the object definition containing the attribute);
- The valid tests (for example, valid value set matching) that may be performed on it; and
- The specific set of error conditions that may cause an object defined failure as a result of performing an attribute-oriented operation.

Attributes themselves do not have accessible properties or subtypes. Attribute values may be explicitly established by external means or by internal means (for example, derived by computation using the values of other attributes).

Attributes shall have a data type that is a standard scalar type defined by this standard. Attributes may have a data type that is a standard data structure defined by this standard. Up

to two indices are available to address constructs of standard types defined by this standard. The valid range for an index is 0 to 32767.

NOTE 3 For example, access to an individual element of a singly-dimensioned array of standard scalar types is supported. As another example, access to an individual element of a data structure contained in a singly-dimensioned array of such structures is supported.

For an attribute constructed as an array, the array size shall be fixed and all elements of an array shall be homogeneously sized. For example, an array of octet strings shall all have the same size octet string for each element in the array. When it is necessary to indicate both the current and maximum dimension of an array, metadata information should be used. This metadata information is described by data type code 406 (Meta\_Data\_Attribute data structure) which is defined in 6.2.6.3 of this standard.

### **12.6.2 Attributes of standard objects**

For each standard object, standard attributes are defined. Each standard attribute has a standard attribute identifier that is used to address the attribute.

Standard object attributes may be extended in the following ways:

- Industry-specific attributes may be added to standard objects.
- Vendor-specific attributes may be added to standard objects.

Extensions to standard attributes need to be coordinated to ensure that attribute identifiers remain unique within an object type. The mechanisms used by industries and vendors to extend the attributes of the standard object are outside the scope of this standard.

### **12.6.3 Attribute classification**

Attributes are classified to provide guidance regarding their change frequency. This information is useful, for example, to gateway devices that cache information. Attribute change frequency is characterized as being:

- Constant;
- Static;
- Static-volatile;
- Dynamic; or
- Non-bufferable.

A constant attribute is unchanging throughout time. An example of a constant attribute is the serial number of a wireless device. Default values in this standard are all constant attributes. The values of these attributes shall be preserved when a device undergoes a warm restart / power-fail, when a device resets to factory defaults, or when a reset command to the relevant attribute or management object is received.

Constant information may be either:

- Fixed information that never changes, such as information to indicate a manufacturer or serial number; and
- Information that does not change during normal system operation, but that may change, for example, when a firmware download occurs.

A static attribute changes its value infrequently. Typically, static data is changed as the result of an external message, request, or event. Some static information may, for example, only be changed by a configuration tool. Operating ranges, units, communication end points, alarms, and constant input values are examples of static information. Attributes storing provisioning information, as well as configuration information provided to a device, are static attributes. The values of these attributes shall be preserved when a device undergoes a warm restart / power-fail or when a reset command to an un-related attribute or management object is received.

A static-volatile attribute changes its value infrequently. Typically, static-volatile data is changed as the result of an external message, request, or event. Some static-volatile

information may, for example, only be changed by a configuration tool. The values of these attributes need not be preserved when a device undergoes a warm restart / power-fail. The values of these attributes shall be preserved when a reset command to an un-related attribute or management object is received.

A dynamic attribute may be changed spontaneously by the device containing the object and without external stimulation from the wireless network. Examples of dynamic attributes are frequently changing values such as process variables, calculations, and timers. Dynamic attributes may be treated differently by a gateway cache infrequently changing (static) values; this is entirely up to gateway internal implementation. A dynamic attribute is not required to survive when a device goes through a warm restart / power-fail or when a device resets to factory defaults. It can be reset when a reset command to the relevant attribute or management object is received.

Non-bufferable information is never buffered; for example, it may be utilized for critical information such as safety information, and for values that change too often to make caching a valid technique. Whenever the value of a non-bufferable attribute is requested, it shall be retrieved from the end device that owns the object and attribute.

NOTE If device local caching is needed, it is the local responsibility of the application.

#### 12.6.4 Attribute accessibility

Network accessible attributes may be:

- Accessible to be read only; or
- Accessible to be both read and written.

#### 12.7 Method model

Methods represent the set of object type-specific interfaces (functions) that can be used to access an object instance. For example, the UploadDownload object supports a StartUpload method.

The standard object methods shall always be available, and shall not be enhanced beyond the definition given by this standard.

Methods shall not be defined if the equivalent result can be achieved using a standard (object type-independent) AL service, for example, the ASL-provided read service. Definition of a method may be warranted, for example, to replace a sequence of communication transactions in order to save energy. Definition of a method may also be warranted when synchronization issues may result if individual actions are used rather than an atomic transaction set.

Standard object methods may be added in future by this standard.

NOTE Time-based triggering of application process activities is not a communication subsystem responsibility. If such time-based triggering is necessary, either a parameter of a method or a dedicated attribute of an object may be employed. If coordination across objects is required, an application object may be defined with an attribute representing the coordinated action, acting as a proxy for the coordination.

#### 12.8 Alert model

The term alert is used to describe an application message that advises or warns the recipient of the presence of an impending or existing situation of interest. The alert model describes alerts reported by application process resident objects and the mechanism to report them.

Two types of alerts are supported, alarms and events. Event is the term used to represent a stateless condition (that is, to indicate a situation has occurred). Events simply report that something happened. An alarm is a stateful condition of an existing situation, for example, that an alarm has transitioned to an abnormal state, or has returned to normal from an abnormal state. The alarm condition remains true until the alarm condition clears. Alert is the term used to describe the messaging of an event condition or alarm condition. Both alarms and events are reported through the alert reporting mechanism defined in this standard.

An alarm is characterized by a state, and alerts are used to report:

- The occurrence of a condition; and
- The return to normal of the previously reported condition.

Events and alarms supported by this standard fall into one of the following categories:

- Device-related;
- Communication-related;
- Security-related; or
- Control process-related.

Each alarm and event defined for an object shall have an associated attribute that describes how it is reported. This associated attribute shall include:

- Whether it is enabled or disabled for reporting; and
- Its priority (importance).

For all alarms, descriptive information shall also include, if the alert is an alarm, whether or not the condition is in or out of alarm.

An analog alarm occurs when a value meets or exceeds an established limit. For analog value alarms, descriptive information shall also include limit information, if any, relating to when the alarm condition is triggered.

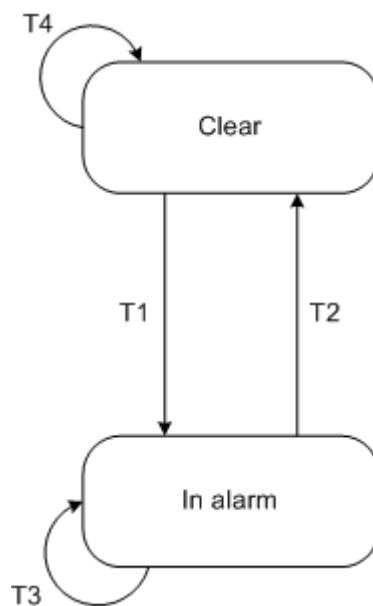
## 12.9 Alarm state model

Table 238 and Figure 111 represent the alarm state model.

**Table 238 – State table for alarm transitions**

Transition	Current State	Event(s)	Action(s)	Next State
T1	Clear	Alarm is detected	Report alarm to ARMO in DMAP	In alarm
T2	In alarm	Clear is detected	Report alarm clear to ARMO in DMAP	Clear
T3	In alarm	Recovery requested	Report alarm to ARMO in DMAP	In alarm
T4	Clear	Recovery requested	Ignore	Clear

NOTE 1 Recovery is typically requested by a remote device.



**Figure 111 – Alarm state model**

Alarm detection applies both to analog and discrete values. Examples of analog alarms include:

- Analog limit alarms (for example, when a value exceeds a high or low threshold);
- Analog deviation alarms (for example, the difference between a process variable and set point);
- A Boolean alarm (for example, when the state of the Boolean matches the discrete limit parameter); and
- Diagnostics, such as those defined in the NAMUR 107 recommendation.

NOTE 1 Alarms that depend upon evaluation of a combination of device-, inter-object-, or intra-object-specific state conditions are considered a local matter and are thus outside the scope of this standard.

NOTE 2 Different levels of alarm conditions are indicated by different alarms. For example, for an analog input, a High alarm represents one level, and a High-High alarm represents a higher level.

## 12.10 Event state model

### 12.10.1 General

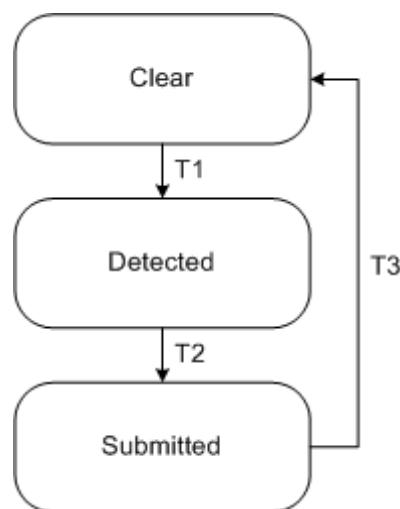
The state model for an event is a subset of the state model for an alarm.

### 12.10.2 State table and transitions

Table 239 and Figure 112 represent the event state table and transitions, respectively.

**Table 239 – State table for event transitions**

Transition	Current state	Event(s)	Action(s)	Next state
T1	Clear	Event condition is detected	Determine report characteristics (e.g., priority)	Detected
T2	Detected	Event condition is reported	Report event to ARMO in DMAP	Submitted
T3	Submitted	Event condition submission to ARMO as completed	Reset to prepare for next event report	Clear



**Figure 112 – Event model**

## 12.11 Alert reporting

### 12.11.1 General

Alerts are reported promptly and accurately time-stamped using a queued unidirectional alert report communication. Queued unidirectional alert reporting involves the alert detecting device reporting the condition using a source/sink communication flow. A queued unidirectional alert acknowledgement is received in return.

NOTE 1 In a published message, status information may also be used to indicate that an alert is available in the reporting device, which may then be accessed using a client/server read service. This method is sometimes used for factory automation; other factory automation systems publish a tag to a server that generates alarms by testing limit values in the server.

NOTE 2 Source/sink communication is used rather than client/server because alert reports may be in a multi-cast in future release of this standard.

### **12.11.2 Alert types**

The alert reporting management object (ARMO) contained in the DMAP provides encapsulation of the alert report, handles timeouts and retries, and throttles alert reporting in a common manner for all the applications contained within the device.

There shall be only one ARMO in each device, in the DMAP. As described in 6.2.7.2.3, the DMAP may provide limited access to entities other than the SMAP in order to support services related to process-related alerts and device-related alerts. Alert acknowledgements shall be addressed to the ARMO.

Diagnostic alerts are specific to the device reporting them. For example, diagnostic alerts may indicate that:

- An error has occurred.
- A symptom has been detected that may indicate that an undiagnosed error occurred.
- A symptom has been detected that may indicate that an error may occur in the future.
- An error will occur if preventative action is not taken.

Diagnostic alerts may pertain to a device as a whole, to an individual component, or to a defined set of components of a device. Diagnostic alerts may be stateless or state-oriented. Diagnostic alerts may be specified by this standard, such as for communication-related alarms, or may be vendor-specific. Diagnostic alerts provide information that can later be examined to establish device and/or communication system behavior patterns.

Process alerts are specific to the process being controlled by the device reporting the process alarm. A process alarm indicates a situation in which the alarmed variable has exceeded established operational limits. For example, a process alert may be generated when a measurable control condition occurs that is outside of desired control system operation parameters.

Process alerts provide information that can later be examined to establish control system behavior patterns, such as:

- Alerts that often occur in a particular sequence;
- Alerts that often occur close in time;
- Alerts that were active for significant periods of time;
- Actions that are required to resolve an alert situation;
- Assistance in determining optimal trip point and hysteresis settings; or
- Information regarding control system performance in terms of alert prevention and resolution.

Process alerts pertain to a particular control object and attribute value of that object (e.g., the PV of an analog input object). Process alerts are typically state-oriented (i.e., alarms).

### **12.11.3 Alert report information**

The APDU header indicates the application and object that initiated the communication. For alert reports, this would indicate the DMAP and the ARMO. The individual alert report information therefore also shall identify the application process and the object within it that is the detecting source of the alert. Additionally, alert reports shall include the following information:

- Network time of detection;
- Individual alert identifier (so that duplicates may be detected by the UAL process);
- Alert class (alarm or event);

- Direction (into alarm, or not an alarm condition (i.e., event, or return to normal));
- Alert category (device diagnostic, communication-related, security-related, or process-related);
- Alert type (object-specific, and within the alert category);
- Process-related category;
- Communication-related category (see 6.2.7.2 for further information on communication alerts);
- Device diagnostic-related category (this standard recommends that device diagnostics be defined for compatibility with the NAMUR recommendation; that is, such diagnostics should indicate if the condition is a failure, off-specification, diagnostic maintenance, or diagnostic check function.)
- Security-related (see Clause 6.1.2 for further information on security alerts).

One octet is used in the coding of alert type information. For all objects, three standard ranges are identified for an enumeration of an object specific alert type. The first range, enumeration ordinal values 0 through 49, defines the values of the enumeration to report alerts defined by this standard. The second range, enumeration ordinal values 51 through 100, is reserved for future use by standard profiles. The third range, enumeration ordinal values 101 through 255, is available for vendor-specific alerts.

- Alert priority (ranges are defined for high, medium, low, and journal-only alert priorities);
- Value size in octets; and
- Value associated with the alert condition.

#### **12.11.4 Alarm state recovery**

If a loss of communication with a wireless device occurs, process industries require that existing conditions be reliably recoverable by an alert receiving device, such as by determining state when an alert receiving device starts up.

NOTE 1 It is possible for multiple alarm conditions to exist simultaneously within a process control device.

Recovery of alarm state may be requested from the ARMO. A single alarm recovery request triggers the re-reporting of all existing alarm conditions in the device of a given category. When recovering alarms, the alarm reporting device shall provide alerts to indicate when the recovery is commencing and when the recovery has completed.

NOTE 2 As event conditions are stateless, they are not recoverable.

### **12.12 Communication interaction model**

#### **12.12.1 General**

Native communication in this standard supports both native protocol and encapsulation of legacy protocols via tunneling. The following types of communication flows are anticipated for compliant devices:

- Queued unidirectional communication (e.g., alarm reporting or alarm acknowledgement);
- Queued bidirectional communication (e.g., read, write, method invocation); and
- Buffered unidirectional publication communication (e.g., publish).

The actual location of the buffers used to hold the data for scheduled unidirectional publication communication is a device local matter.

NOTE Buffered scheduled data publication (periodic, change of state, and application driven publication) all occur (as needed) within the scheduled phase. Communication contracts indicating the need for periodic communication employ buffered unidirectional publication communication. Communication contracts for aperiodic communication employ a queued communication paradigm.

### **12.12.2 Buffered unidirectional publication communication**

#### **12.12.2.1 General**

Buffered unidirectional communication is used when a publishing application is sending a message to a subscribing application. The buffer contains the message to be sent. On each buffered unidirectional publication contract, there is a parameter to indicate if the buffer is to always be transmitted (whether the content has been updated or not), or if the buffer is only to be transmitted if it has been updated since the prior transmission.

#### **12.12.2.2 Buffer content always transmitted**

In buffered unidirectional communication, if a publishing communication protocol suite receives another ASL publish service request for a particular communication contract before the previous message has been transmitted, the new request replaces the previous request. In the subscriber, if a new message is received before the previous one has been delivered to the application, the new message shall replace the previous undelivered message.

NOTE 1 In establishing a contract for periodic communication, the system manager ensures that there is adequate capacity within the intermediate devices along a route to support the periodic communication.

NOTE 2 If an application receives an older publication received after a newer one, an application may discard the older publication.

If a publishing communication protocol suite does not receive a new service request for the contract when it is time to transmit, the previous request shall be retransmitted. If the subscriber receives the same application service request in succession, the subscribing application shall treat this situation as receipt of a duplicate message. Application handling of a duplicate buffered message is left to the application, and is not defined by this standard.

#### **12.12.2.3 Buffer content transmitted on change only**

This mode of buffered communication supports a change of state communication mechanism.

In buffered unidirectional communication, if a publishing communication protocol suite receives another service request for a particular communication contract before the previous message has been transmitted, the new request replaces the previous request. In the subscriber, if a new message is received before the previous one has been delivered to the application, the new message shall replace the previous undelivered message.

NOTE In establishing a contract for periodic communication, the system manager ensures that there is adequate capacity within the intermediate devices along a route to support the periodic communication.

If a publishing communication protocol suite does not receive a new service request for the contract when it is time to transmit, the previous request shall not be retransmitted. If the subscriber receives the same application service request in succession, the subscribing application shall treat this situation as an error situation. Application handling of a duplicate buffered message is left to the application, and is not defined by this standard.

### **12.12.3 Queued unidirectional communication**

Queued unidirectional communication supports queued distribution of unconfirmed (unidirectional) ASL communication services. To satisfy this type of communication need, the lower layers of the correspondents are expected to provide a queued data transfer service.

Application handling of a duplicate AlertReport shall result in sending another AlertAcknowledgement. Receipt of a duplicate AlertAcknowledgement shall be ignored. Application handling of duplicate queued unidirectional Tunnel messages is left to the application, and is not defined by this standard.

### **12.12.4 Queued bidirectional communication**

#### **12.12.4.1 General**

Queued bidirectional communication supports queued distribution of confirmed (bidirectional) ASL communication services. To satisfy this type of communication need, lower layers of the correspondents are expected to provide a queued data transfer service.

This maximum number of simultaneously outstanding queued bidirectional (client/server) confirmed service requests permitted for a contract is indicated to the application process by

the communication contract when the contract is granted. The default value for this maximum value shall be 1, i.e., the default indicates that contract supports only one outstanding request at any given time.

Application handling of a duplicate request is to send another response. Application handling of a duplicate response when a response is received that does not match a pending request identifier shall result in ignoring the response.

#### **12.12.4.2 Retries and flow control**

##### **12.12.4.2.1 General**

The application layer defined by this standard is required to track what happens to queued service client/server requests that it sends. This is necessary for two reasons, to ensure reliability of delivery and for flow control.

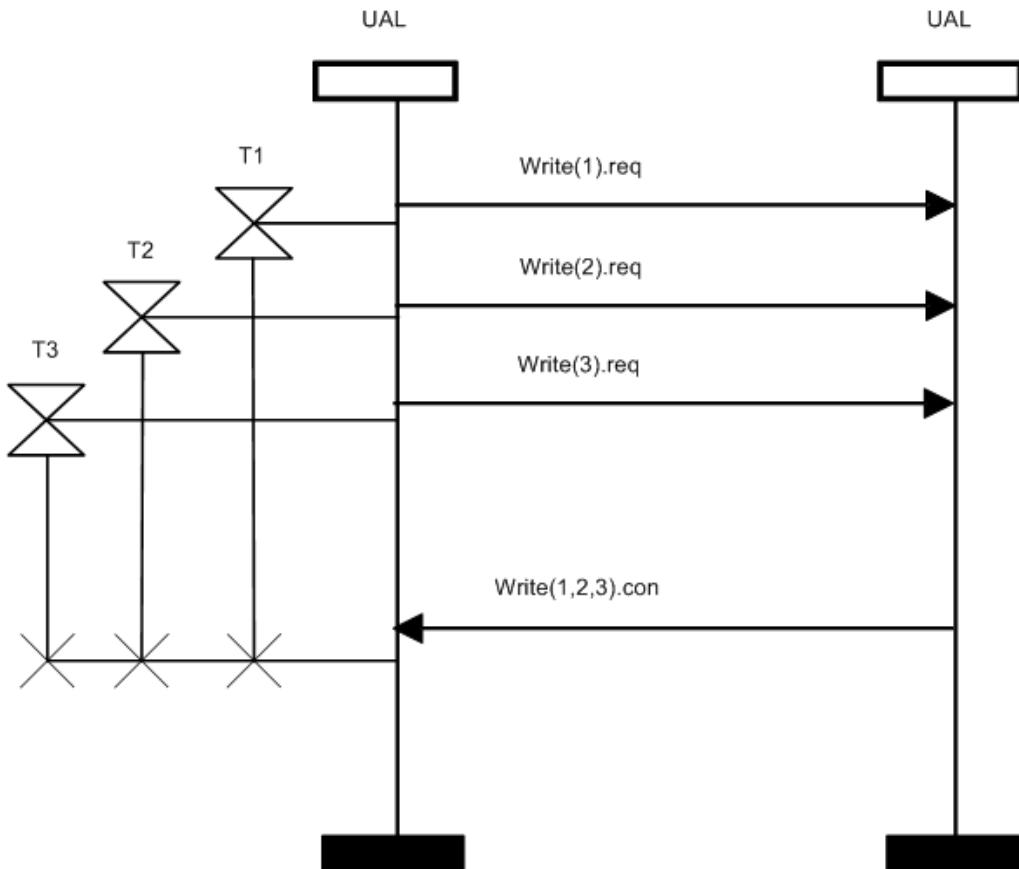
For delivery reliability, the application needs to be able to determine when it should re-send (retry) its message. There are two situations in which message retry may be necessary, first when the message request did not arrive at its final destination, and second when the message request arrived, but the application response did not make it back to the original requestor. Flow control is necessary to ensure that the destination device is not overwhelmed with messages it cannot handle, as well as to protect the network and optimize network throughput.

This standard supports both forward explicit congestion notification by the lower communication protocol suite, and supports an application level echo back to a client of four-part service requestor if congestion occurred on the path taken for the original service request.

To enable multiple outstanding requests simultaneously while still allowing an application to achieve both reliable delivery and flow control, each client request shall contain a unique identifier. Retransmission of a request (retries) shall use the same identifier. This identifier enables the application to implement a sliding window technique to control flow. The client shall start a service related time-out timer when it initiates a client service request. This timer shall be based on round trip times (RTT) for messages, and shall allow sufficient time for a message from an application in device X to reach the destination application in device Y, for the server application in device Y to issue a response, and for the response to travel back to the service requesting application in device X.

NOTE This method is commonly known in communications as communication using positive acknowledgment with retransmission, or simply PAR.

Figure 113 represents an example of three simultaneously outstanding write request messages, with a single concatenated message that contains the responses for all the outstanding write requests. Concatenation is used in order to save messaging traffic.



**Figure 113 – A successful example of multiple outstanding requests, with response concatenation**

#### 12.12.4.2.2 Retries and timeout intervals

##### 12.12.4.2.2.1 General

The method defined by IETF RFC 2988 shall be used for calculating an appropriate value for the retry timer-out interval (RTO). To compute the current RTO, a client shall maintain two state variables, SRTT (smoothed round-trip time) and RTTV (round-trip time variation), within the scope of a contract.

Until a round-trip time (RTT) measurement has been made for a segment sent between the client and server, the client should set RTO = 3 s.

When the first RTT measurement R is made, the client shall set:

- SRTT = R
- RTTV = R/2
- RTO = SRTT + 4\*RTTV

When a subsequent RTT is available, R is made;, the client shall update the RTTV, SRTT, and RTO using the following calculations, where the recommended value for  $\beta = 0,25$ , and the recommended value for  $\alpha = 0,125$ :

- RTTV =  $(1 - \beta) * RTTV + \beta * |SRTT - R|$
- SRTT =  $(1 - \alpha) * SRTT + \alpha * R$
- RTO = SRTT + 4\*RTTV

Whenever RTO is computed, the RTO shall be rounded based on the following rules:

- If RTO < 1 s, set RTO = 1 s.
- If RTO > 60 s, set RTO = 60 s.

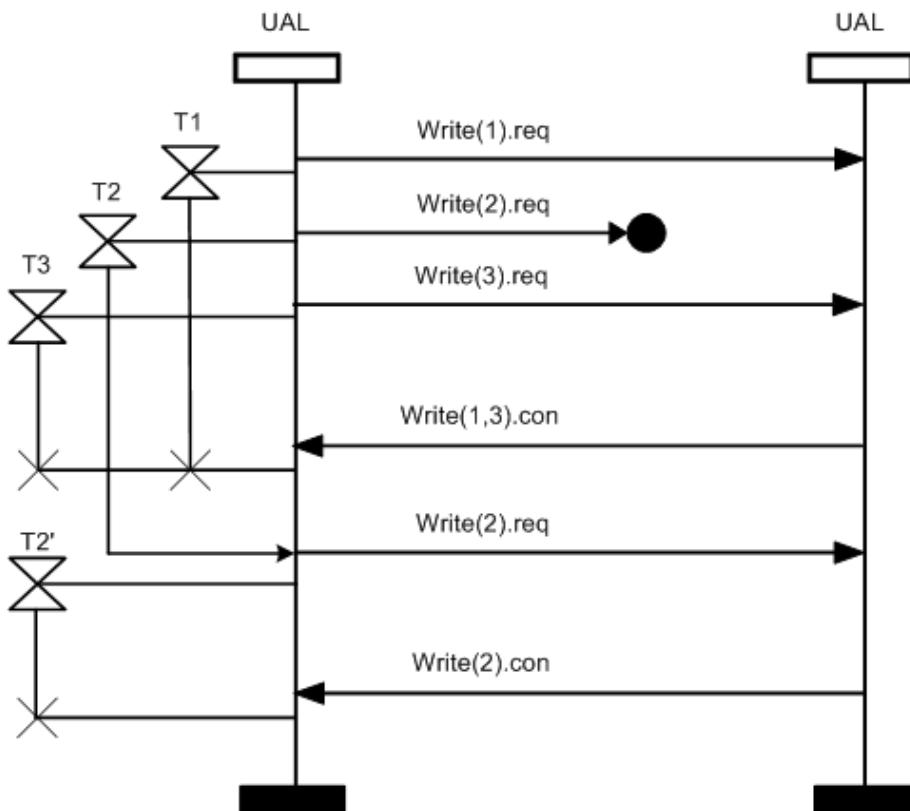
Determination of timeout occurrence is a local matter. When a timeout has been determined to have occurred, exponential backoff shall be employed for consecutive timeouts by setting  $RTO = RTO * 2$  to send the retries. The maximum value of 60 s should be used to provide an upper bound to this doubling operation. Retries cease either when a response is received, or when the maximum retry limit is reached. The maximum retry permitted for a client request is indicated via an attribute of the UAP management object. **The value selected for the maximum retry permitted is a local matter.**

NOTE IETF RFC 2988 contains a recommendation regarding management of the TimeoutInterval timer.

#### 12.12.4.2.2.2 Retries for unordered messages

Unordered messages are independent in that the order of responses may be received in a different order than the order in which the requests were sent. Accordingly, each request message times out and is retried independently.

Figure 114 is an example of how a timeout and retry of the second message in a sequence of three unordered messages, due to failure of the request to reach the server, is handled.

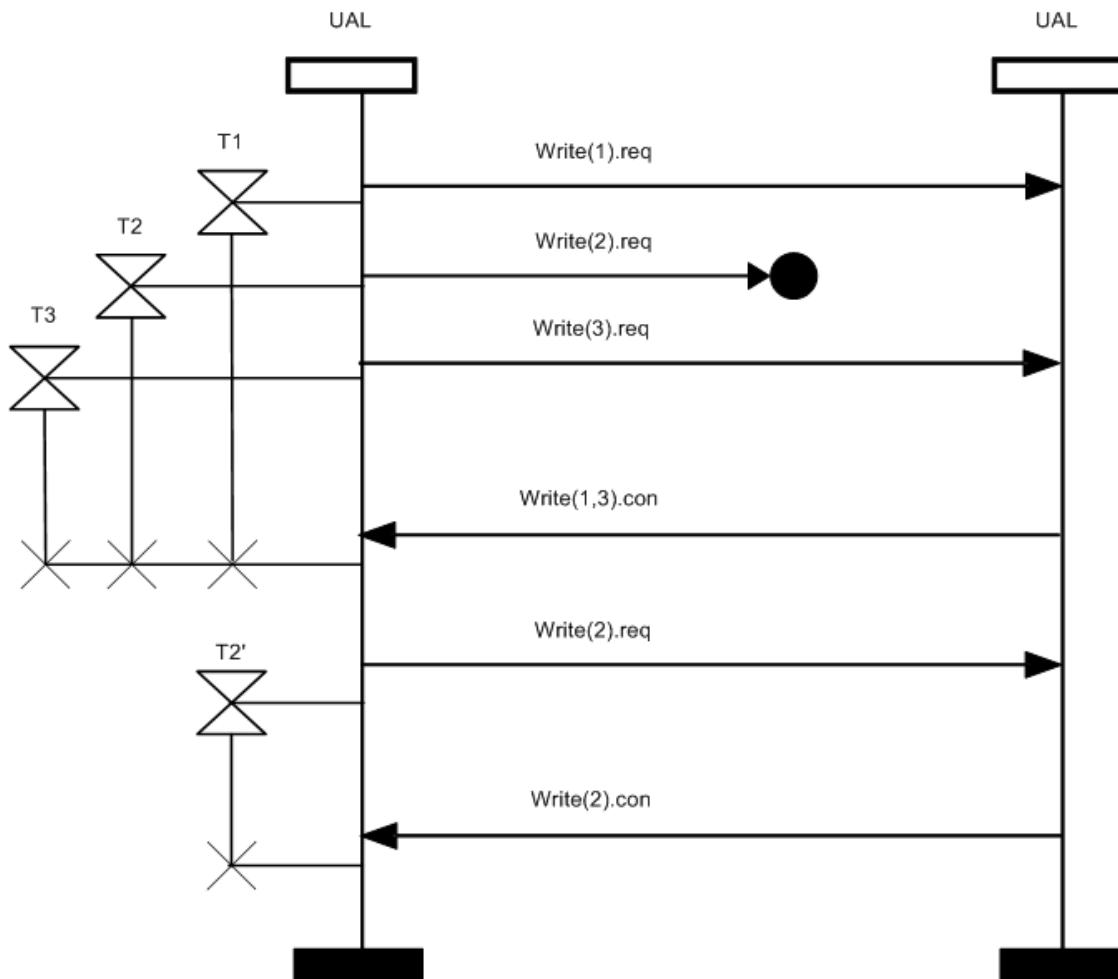


**Figure 114 – An example of multiple outstanding unordered requests, with second write request initially unsuccessful**

#### 12.12.4.2.2.3 Retries for ordered messages

Ordered messages are order-dependent; that is, the order of responses may not be received in a different order than the order in which the requests were sent. Accordingly, if a later message receives a response before an earlier message, it indicates that the message for which no response was received shall be timed-out and retried.

Figure 115 is an example of how a timeout and retry of the second message in a sequence of three ordered messages, due to failure of the request to reach the server, is handled.



**Figure 115 – An example of multiple outstanding ordered requests, with second write request initially unsuccessful**

Ordered delivery only pertains to upload/download. The Max\_Send\_Window\_Size for upload/download communication contracts shall be fixed at 1. As such, ordered message delivery is not supported by the lower layers of the protocol suite defined by this standard.

#### 12.12.4.2.3 Flow control

C/S communications are not application process rate controlled; rather, application layer flow rate fairness is enforced by the application processes on a per contract basis, in order to minimize the cost fairness of congestion on the network.

Max\_Send\_Window\_Size (Table 26) for a contract is the maximum number of client requests that may be simultaneously awaiting a response within the scope of a contract. It is recommended that clients use sequentially contiguous request identifiers. The value of Max\_Send\_Window\_Size is established on a contract basis by the system manager. The OutstandingList represents the messages that have been sent, and that are currently awaiting a response. The AvailableSendWindowSize represents the usable send window, that is, the set of client requests that may be sent, without violating the Max\_Send\_Window\_Size, taking into consideration the number of messages contained in the OutstandingList. When the windows are empty, and CurrentSendWindowSize equals the Max\_Send\_Window\_Size, the usable send window stretches from the last acknowledged client request for the next Max\_Send\_Window\_Size number of requests, and represents the set of client requests that may be sent, without violating the Max\_Send\_Window\_Size.

Applications shall initially set their value for their CurrentSendWindowSize to be one (1). RTT is then measured by the application for n complete transactions. If no timeout occurs during these transactions, and if the Max\_Send\_Window\_Size has not been reached, the

`CurrentSendWindowSize` shall be incremented by one (1). The value of `CurrentSendWindowSize` shall be equal the size of the `CurrentSentWindowLimit` +1.

NOTE Requiring the number of transactions equal to the window size to succeed before increasing the size of the window is better (more conservative) than TCP congestion avoidance requires.

As a response is received, the corresponding request moves from the sent window to the response completed window, and the size of `AvailableSendWindowSize` increases by one if the `Max_Send_Window_Size` has not been reached. If a timeout occurs awaiting a response, message loss or congestion is indicated. If this occurs, then:

- No additional message shall be placed into the `OutstandingList` until after the `OutstandingList` has first become empty.
- The `CurrentSendWindowLimit` shall be set to one (1).
- The messages that were in the `OutstandingList` at time of collapse shall be retried in order, according to the retry policies defined above. Retries use exponential backoff if the first retry does not succeed. Retries shall continue until either a response is received or the maximum number of retries has been met. When either of these conditions occurs, the message handling is considered complete, and the message shall be removed from the `OutstandingList`.

Client requests may continue again, building up the `CurrentSendWindowLimit` to the `Max_Send_Window_Size` value using the procedure described above.

EXAMPLE In an example of how the windows are used in a situation when no retries are required:

Let `Max_Send_Window_Size` be a constant, equal to the maximum number of simultaneously outstanding requests that may exist for the contract. This limit is established by the system manager.

Let `CurrentSendWindowSize` be the variable that represents the number of simultaneously outstanding requests that may exist for the contract at point in time t. `CurrentSendWindowSize` is a non-negative integer that may be less than or equal to `Max_Send_Window_Size`.

Let `UsedSendWindowSize` be the variable that represents the number of simultaneously outstanding requests that are still awaiting responses.

`AvailableSendWindowSize` = `CurrentSendWindowSize` – `UsedSendWindowSize`.

Assume: `Max_Send_Window_Size` = 3

T1: (Initialization): `CurrentSendWindowSize` = 1; `UsedSendWindowSize` = 0, `AvailableSendWindowSize` = 1

T2: Message  $M_1$  sent: `CurrentSendWindowSize` = 1; `UsedSendWindowSize` = 1, `AvailableSendWindowSize` = 0

T3: Message  $M_1$  response received: `CurrentSendWindowSize` = 1; `UsedSendWindowSize` = 0, `AvailableSendWindowSize` = 1

T4: Message  $M_2$  sent: `CurrentSendWindowSize` = 1; `UsedSendWindowSize` = 1, `AvailableSendWindowSize` = 0

T5: Message  $M_3$  response received: `Current SendWindowSize` = 2; `UsedSendWindowSize` = 0, `AvailableSendWindowSize` = 2

The `CurrentSendWindowSize` has been incremented by one, since:

(a) it has been at size 1, and 2 transactions have completed successfully.

(b) `CurrentSendWindowSize` < `Max_Send_Window_Size`.

T6: Message  $M_4$  sent: `CurrentSendWindowSize` = 2; `UsedSendWindowSize` = 1, `AvailableSendWindowSize` = 1

T7: Message  $M_5$  sent: `CurrentSendWindowSize` = 2; `UsedSendWindowSize` = 2, `AvailableSendWindowSize` = 0

T8 Message  $M_4$  and  $M_5$  responses received : `CurrentSendWindowSize` = 2; `UsedSendWindowSize` = 0; `AvailableSendWindowSize` = 2

T9: Message  $M_6$  sent: `CurrentSendWindowSize` = 2; `UsedSendWindowSize` = 1; `AvailableSendWindowSize` = 1

T10: Message  $M_7$  sent: `Current SendWindowSize` = 2; `UsedSendWindowSize` = 2; `AvailableSendWindowSize` = 0

T11: Message  $M_6$  response received : `Current Send Window Size` = 3; `UsedSendWindowSize` = 1;

`Available SendWindowSize` = 2

The `CurrentSendWindowSize` has been incremented by one, since:

(a) it has been at size 2, and 3 transactions have completed successfully.

(b) `CurrentSendWindowSize` < `Max_Send_Window_Size`.

T12: Message  $M_8$  sent: `CurrentSend Window Size` = 3; `UsedSendWindowSize` = 2; `Available SendWindowSize` = 1

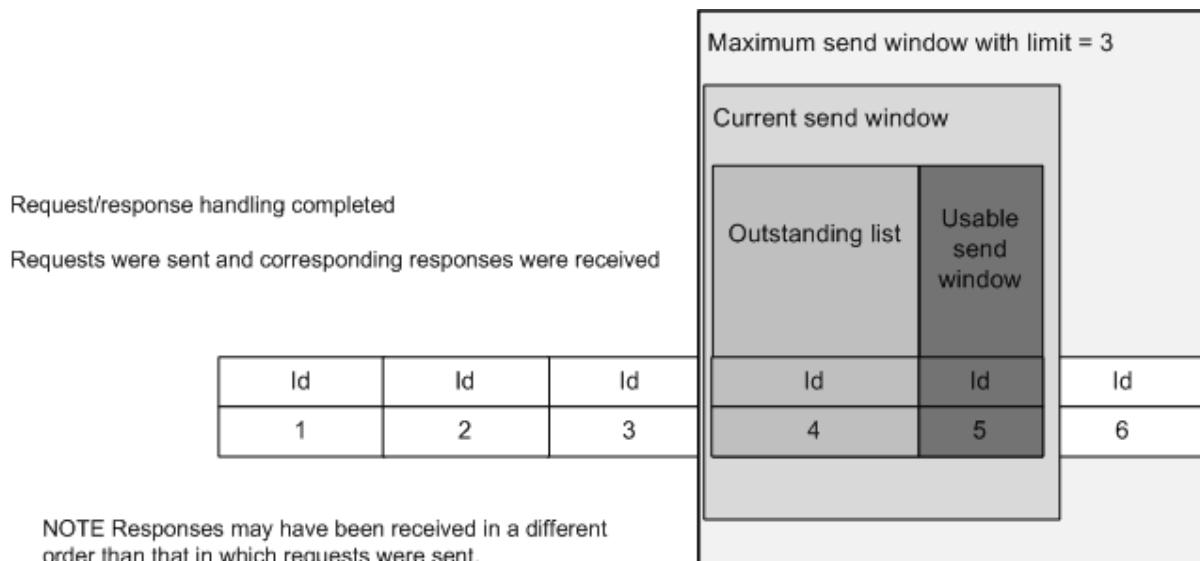
T13: Message  $M_9$  sent: `CurrentSendWindowSize` = 3; `UsedSendWindowSize` = 3; `AvailableSendWindowSize` = 0

T14: Messages M<sub>7</sub> response received: CurrentSendWindowSize = 3; UsedSendWindowSize = 2; AvailableSendWindowSize = 1

T15: Message M<sub>10</sub> sent: CurrentSendWindowSize = 3; UsedSendWindowSize = 3; AvailableSendWindowSize = 0

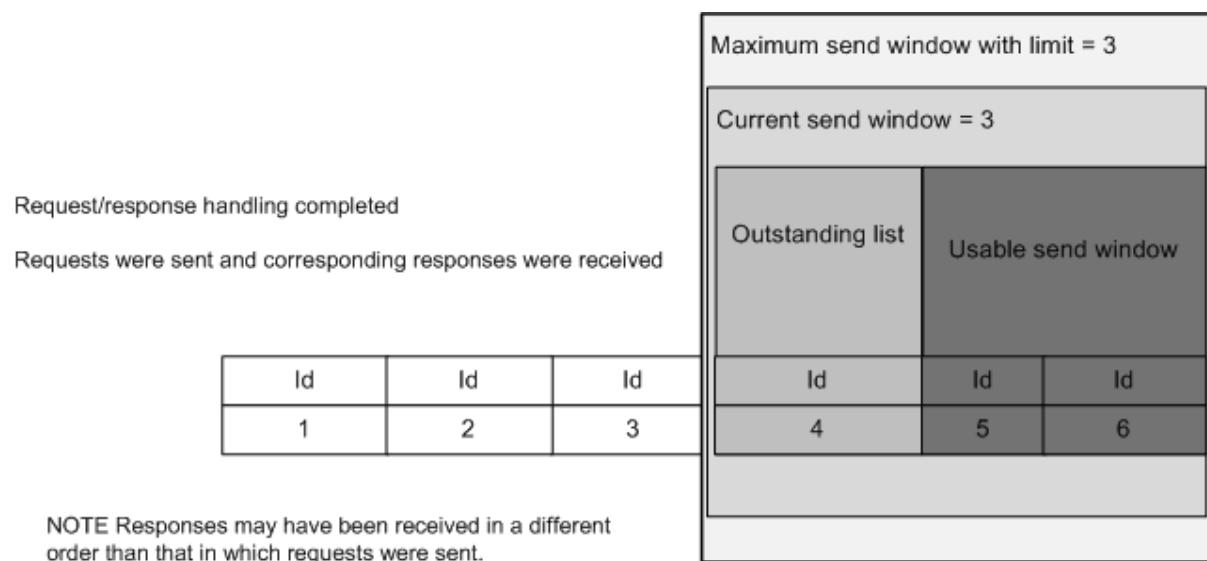
...

Figure 116 depicts a situation wherein the current send window has not yet built up to the maximum send window limit size. In this example, the maximum send window limit is three messages, one message in the outstanding list has been sent and is awaiting a response, and one message may be sent before the usable send window limit is reached.



**Figure 116 – Send window example 1, with current send window smaller than maximum send window**

Figure 117 depicts a situation wherein the current send window has built up to the maximum send window limit size. In this example, the maximum send window limit is three messages, one message in the outstanding list has been sent and is awaiting a response, and two messages may be sent before the usable send window limit is reached.



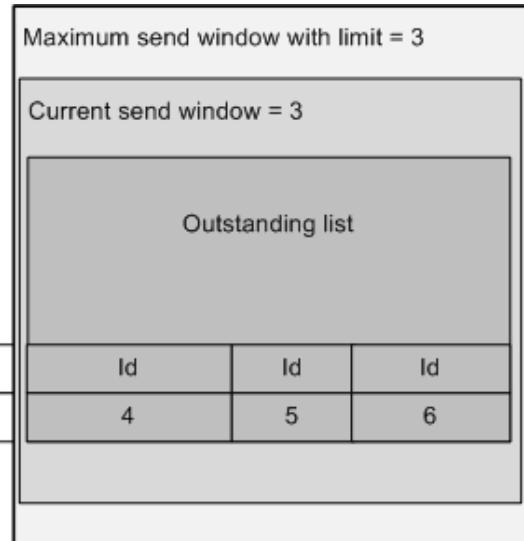
**Figure 117 – Send window example 2, with current send window the same size as maximum send window, and non-zero usable send window width**

Figure 118 depicts a situation wherein the current send window has built up to the maximum send window limit size. In this example, the maximum send window limit is three messages, and three messages have been sent and are awaiting responses.

Request/response handling completed  
 Requests were sent and corresponding responses were received

Id	Id	Id
1	2	3

NOTE Responses may have been received in a different order than that in which requests were sent.



**Figure 118 – Send window example 3, with current send window the same size as maximum send window, and usable send window width of zero (0)**

#### 12.12.4.2.4 Probing for congestion

Some system configurations are more likely than others to incur message loss due to network congestion. In system configurations where congestion is more likely, an application may wish to regulate its AL service requests based on whether or not network congestion is present. To do this, an application may probe for congestion. To effect such a probe, the application may engage in a simple single message exchange.

NOTE 1 Probing is intended for diagnostic purposes only. A single message is used to ensure that the probes do not overload the network, and to ensure that the response to a probe can be distinguished.

The message request to use when probing shall be a non-concatenated read service. The read request and corresponding read response for the probe shall each fit within a single DL fragment. Any object attribute may be used as a probe; however, it is recommended that the same object and attribute be used consistently for probing. For example, a standard attribute of the UAPMO may be used to probe user application processes, and a standard attribute of the DMAP DMO for probing the DMAP since those objects are required to be present in the corresponding applications.

If the probe timeout does not expire prior to reception of the response, then the application should assume that there is no congestion. However, if the response does not return before the retry timeout interval passes, this indicates a higher probability that network congestion is present. In this situation, the application process shall self-regulate its communication activities by setting its CurrentWindowSize to 1. See 12.12.4.2.3. If the application desires to send another congestion probe message, it may do so using exponential backoff as described in 12.12.4.2.2.1, but shall use a different request identifier for each message probe.

NOTE 2 This permits an application to compute RTT specific to congestion probing, and to make congestion decisions accordingly based on the congesting probing RTT data.

For example, a UAP in device X may issue a read service request for the required standard state attribute of the required UAPMO contained within the destination application in device Y. This read request may be treated as an application process-initiated congestion probe.

In the specific case of a download or upload, an application may probe for congestion. In these situations, congesting probing shall be performed as follows:

- To probe prior to commencing a download operation, the client probe shall be a read request to the UploadDownload attribute MaxDownloadSize.
- To probe for congestion during a download operation, the client probe shall be a read service request for the UploadDownload objects LastBlockDownloaded attribute.

- Prior to commencing an upload operation, the client probe shall be a read request to the UploadDownload attribute MaxUploadSize.
- To probe for congestion during an upload operation, the client probe shall be a read service request to the UploadDownload objects LastBlockUploaded attribute.

### 12.12.5 Communication service contract

A UAP makes a contact request to the local DMAP via an UAPME-n SAP in order to establish an agreement for a communication service needed by the UAP. If the need can be met, the DMAP provides a service contract identifier to the UAP that represents the agreement. The contract identifier is passed from the UAP to the ASL when it makes an ASL service request. This service contract ID is then used by the lower communication protocol suite to identify the layer-specific characteristics of the contract that have been established into the lower communication protocol suite layers by the local DMAP as part of establishing the service contract. The communication of information required from the UAP to the DMAP in order to acquire a service contract identifier is a device-internal matter, and hence not specified by this standard.

NOTE 1 Other local services may be provided to a UAP from the DMAP to permit the UAP to take actions, such as to cancel an existing contract.

All communication contracts have a base set of information. Additional required information depends on the type of communication relationships desired. For example, a publish/subscribe relationship for periodic communication requires specification of the desired phase and period.

This standard does not specify how to determine the information needed by the UAP to specify the characteristics of a contract. For example, such information may be configured, such as the periodicity and phase to use in scheduled communication, or such information may be determined by the vendor of the device that contains the UAP.

Contract requests may be negotiated down by the system manager. UAP policies regarding the handling of negotiated down contracts, as well as policies regarding the handling of a declined contract, are outside the scope of this standard. See 6.3.11 for further information about the information that needs to be specified to request a contract.

The publishing period is represented by a signed 16-bit integer value. A positive value indicates publication period as a multiple of 1 s (e.g., a value of 5 indicates a publishing period of 5 s; a value of 3600 indicates a publishing period of 1 hour). A negative value indicates publication on a fraction of 1 s (e.g., -4 indicates publish every  $\frac{1}{4}$  s, -2 indicates publish every  $\frac{1}{2}$  s). A zero value indicates that no publishing should occur.

The periodicity selected should be based on the efficiency of the operation with this standard and typical process practice.

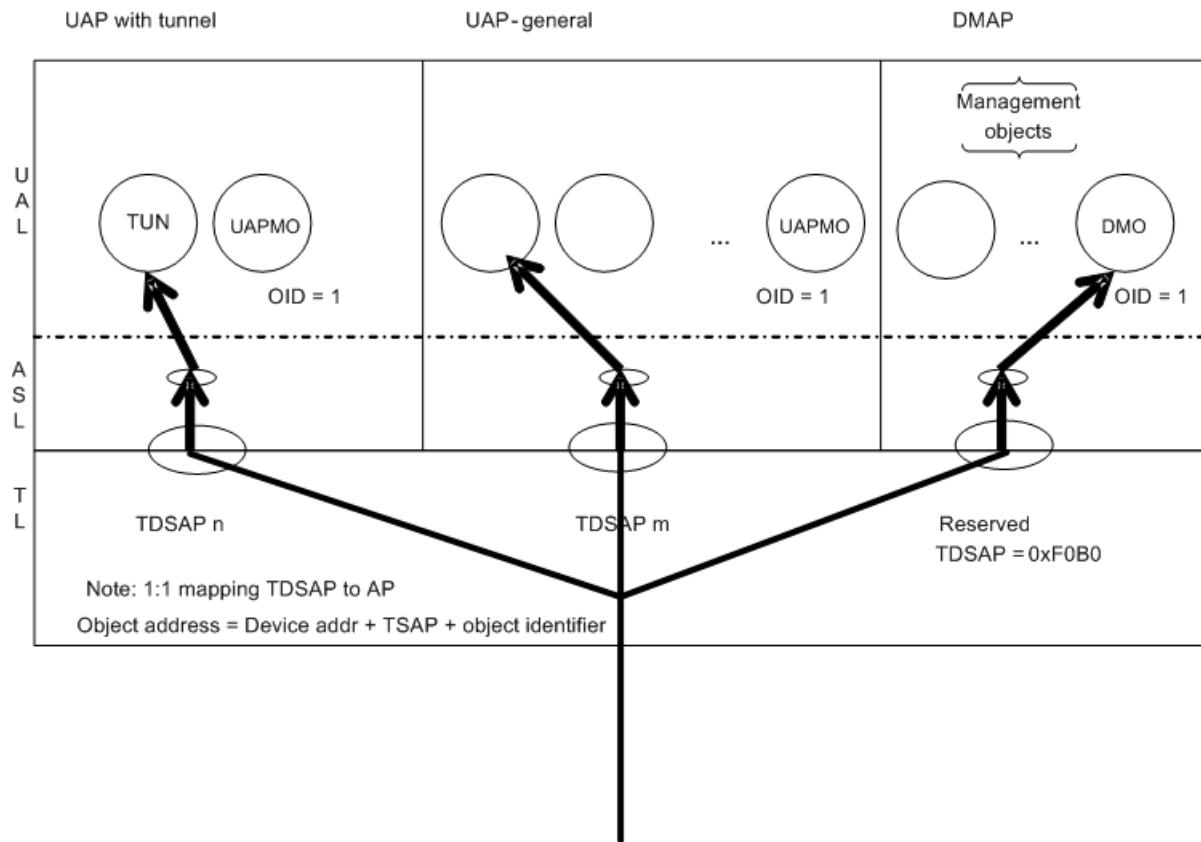
NOTE 2 The DMAP need not know the destination TSAP port in order to create a service contract, as contract establishment is concerned with resources for communication over the network conditions, whereas the destination port is used within the destination device, after the over-the-network communication has occurred.

NOTE 3 Policies to retry establishment of a contract in the event of failure of contract establishment or revocation of a contract are behaviors required of the device as a whole (as opposed to behaviors required of a component within the device). Device-level behaviors are discussed in 6.3.11.2.4.2.

## 12.13 Application layer addressing

### 12.13.1 General

Certain information is required to address an object, an object's attribute, an element of an object's attribute (e.g., an element of a structure or an element of an array), or an object's method in native communications. Figure 119 represents the general addressing model for UAL process objects.



**Figure 119 – General addressing model**

### 12.13.2 Object addressing

An object is addressed in unicast communications by specifying:

- Its containing device physical address.
- The transport layer (TL) data service access point (TDSAP) is used to communicate with the unique UAL process that contains the object (that is, the TDSAP maps 1:1 to an application process). TDSAP number 0 is dedicated for use by a DMAP only. A DMAP is present in every device. TDSAP number 1 is dedicated for use by an SMAP only. Devices that do not have an SMAP present therefore may not use TDSAP number 1.
- A TL port number corresponds to a particular TDSAP. TL ports shall be assigned in consecutive order to TDSAPs, starting from the first available TL port. For example, TDSAP number 0 shall be correlated with the first TL port 0xF0B0; TDSAP number 1 shall be correlated with the second port, 0xF0B1, and so forth. Particular TLSAPs, and their corresponding TL ports, shall be reserved by this standard in order to be well-known by all applications. Specifically, the DMAP in every device shall have the reserved transport port number 0xF0B0, which is associated with TDSAP number 0. The SMAP in a device shall have the reserved transport port number 0xF0B1, which is associated with TDSAP number 1. Devices that do not have an SMAP present shall not use TL port number 0xF0B1. It is recommended that UAPs with the largest messaging use the TL ports numbered 0xF0B2 through 0xF0BF, as they are represented in compressed form over the network. See 11.4.4 for further details on TDSAPs and TL ports.
- The unique object identifier within the UAL process.

In order to minimize the encoding of application messages, it is recommended that object identifiers be allocated consecutively, starting at 1.

NOTE 1 Object identifier 0 is reserved by this standard for the use of the application process management object contained within all application processes.

NOTE 2 Multicast communication is not supported.

### 12.13.3 Object attribute addressing

An attribute of an object is addressed by specifying:

- The addressing of its containing object; and
- The unique attribute identifier within the object.

In order to minimize the encoding of application message attributes, it is recommended that attributes be allocated consecutively, starting at 1.

NOTE Attribute identifier 0 is reserved by this standard for future use.

### 12.13.4 Object attribute element addressing

#### 12.13.4.1 General

Addressing for attributes is defined based on the type of attribute. This standard supports the following attribute types:

- Scalars;
- Standard data structures defined by this standard;
- Singly-dimensioned arrays; and
- Doubly-dimensioned arrays.

#### 12.13.4.2 Scalars

This standard supports access to attributes that are scalar:

- Boolean;
- Integer8;
- Integer16;
- Integer32;
- Unsigned8;
- Unsigned16;
- Unsigned32;
- Unsigned64;
- Unsigned128;
- Float;
- VisibleString;
- OctetString;
- Bitstring; and
- DoublePrecisionFloat.

NOTE See subclause 12.22.3 on data types for the scalar types supported by the standard

#### 12.13.4.3 Standard data structures

This standard supports access to a standard data structure. Supported access to such structures that are not contained within an array is as follows:

- A standard structure may be accessed in its entirety by specifying to access element zero (e.g., a.0).
- A scalar element of a standard structure may be accessed by identifying the attribute and an index that corresponds to the desired element (e.g., a.i, where a is a standard structure and i represents the identifier of a scalar element).

- A standard structure element of a standard structure may be accessed in its entirety (e.g., a.b, where a and b are standard structures, shall access standard structure b within standard structure a in its entirety).
- A scalar element of a standard structure element of a standard structure (e.g., a.b.c, where a and b are standard structures, and c is a scalar supported by this standard).
- A singly-dimensioned array element of a standard structure may be accessed in its entirety (e.g., a.b[0], where a is a standard structure and b is a singly-dimensioned array, and the array b is comprised either of scalars as defined by this standard or standard structures).
- A single element of a singly-dimensioned array of scalars that is an element of a standard structure (e.g., a.b[i], where a is a standard structure, b is a singly-dimensioned array of standard structures, and i represents the scalar element of interest).
- A single element of a singly-dimensioned array of standard structures that is an element of a standard structure (e.g., a.b[i], where a is a standard structure, b is a singly-dimensioned array comprised either of scalars as defined by this standard or standard structures, and i is the element of interest).

#### **12.13.4.4 Singly-dimensioned arrays**

This standard supports access to a singly-dimensioned array of scalars as defined by this standard or standard data structures as follows:

- A single element of a single dimension arrays comprised of scalars (e.g., a[i]).
- A singly-dimensioned array of scalars or standard data structures may be accessed in its entirety (e.g., a[0]).
- An element of a singly-dimensioned array comprised of standard structures (e.g., a[i][0], where a represents the array, i represents the element in the array that is the standard structure, and 0 represents access to the entire structure).
- A scalar element of a standard structure contained in a singly-dimensioned array (e.g., a[i].j, where a represents the structure as defined by this standard, i represents the array element that is the standard structure, and j represents the element within the standard structure).
- A singly-dimensioned array contained as an element of a singly-dimensioned array (e.g., a[i][0], where a is an array of standard structures, j represents an element of the array, and 0 is used to indicate that access to the entire array is desired).
- An element of a singly-dimensioned array of scalars or standard structures contained as an element of a singly-dimensioned array (e.g., a[i][j], where a represents the outer scope array, i represents an element of that array that is itself an array, and j represents the element of the inner scope array).

#### **12.13.4.5 Doubly-dimensioned arrays**

This standard supports access to a doubly-dimensioned array of scalars as defined by this standard or standard data structures as follows:

- A scalar element of a doubly-dimensioned array (e.g., a[i][j]);
- A doubly-dimensioned array in its entirety (e.g., a[0][0]);
- A row of a doubly-dimensioned array (e.g., a[i][0]); or
- A column of a doubly-dimensioned array (e.g., a[0][j]).

NOTE a structured or array element of a doubly-dimensioned array may be accessed in its entirety only by representing it as an octet string (a[i][j]).

#### **12.13.5 Object method addressing**

An objects method is addressed by specifying:

- The addressing of its containing object; and

- The object unique method identifier.

## **12.14 Management objects**

Standard management objects to manage the device as a whole are defined in this standard. These objects are defined by 6.2 and are accessed through a UAL-contained MP that may include, for example, a management object to support identification of the device, management objects for each layer of the communication protocol suite, and a management object to report alerts from the device.

**NOTE** Though each object tracks its own event and alarm conditions, the reporting of such conditions is specified by a single ARMO for the device as a whole. This object manages aspects including, but not limited to, the local alert reporting queue(s), the local timer(s) associated with retransmitting if an individual alert acknowledgement is not received, the local alert queue overflow handling, and requests for alarm recovery. See 6.2.7.2 for further details.

## **12.15 User objects**

### **12.15.1 General**

Standard UAP-containable objects are defined to enable interoperability across industries and segments. These objects may be industry-independent (that is, applicable across industries supported by this standard), or industry-dependent (that is, applicable to a particular industry supported by this standard, but not used across industries).

### **12.15.2 Industry-independent objects**

#### **12.15.2.1 General**

The standard objects (UAPMO, ARO, UDO, Concentrator, Dispersion, Tunnel, and Interface) are applicable across industries supported by this standard.

#### **12.15.2.2 User application process management object**

##### **12.15.2.2.1 General**

There is exactly one addressable UAP management object (UAPMO) per UAP supported by the AL defined by this standard. The numeric object identifier of an object indicates a particular object instance. The numeric object identifier of the UAPMO in every UAP shall be fixed and shall have the value one (1). This object facilitates common management of application processes within a device. Attributes of this object are used to indicate such information as the version/revision of the application process and the logical status of the application process. For example, an attribute of the UAPMO indicates if the corresponding UAP is active or inactive.

**NOTE** 1 It is possible for a UAPMO optionally to support management of a particular group (set) of objects within the UAP.

**NOTE** 2 Dynamic instantiation of UAPs is outside the scope of this standard.

##### **12.15.2.2.2 Object attributes**

A UAPMO has the attributes defined in Table 240.

**Table 240 – UAP management object attributes**

<b>Standard object type name: UAP management object</b>				
<b>Standard object type identifier: 1</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16	N/A
			Classification: Constant	
			Accessibility: N/A	
			Initial default value: 0	
			Valid value set: 0	
UAP_ID	Object key identifier	Associated TLDE SAP	Type: Local matter (as defined by local TL)	Local TDLE-SAP
			Classification: Constant	
			Accessibility: N/A	
			Initial default value: 0 = N/A	
			Valid value set: as defined by TL	
UAP_TL_Port	Object key identifier	Associated TL port	Type: Unsigned16	NOTE 1 The specification of the UAP to its local TL is a local matter.  NOTE 2 Transport defines the hexadecimal value set 0xF0B0+n (where n may range from 0 to 15) to specify the most compressed representation (into 4 bits) for communication.
			Classification: Constant	
			Accessibility: N/A	
			Initial default value: 0 = N/A	
			Valid value set: as defined by TL	
Reserved for future use	0			
VersionRevision	1	VersionRevision of the UAP	Type: VisibleString	Human readable identification associated with the UAP Management object.  NOTE The UAP vendor determines content of this attribute.
			Max length: 64 octets	
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: N/A	
State	2	Status of UAP	Type: Unsigned8	See Table 241
			Classification: Static	
			Accessibility: Read only	
			Initial default value: Active	
			Valid value set: 0=Inactive 1=Active 2=Failed	
Command	3	Command to change the state of the UAP	Type: Unsigned8	The value None shall not be indicated in a write request.  Soft reset shall preserve configuration/commissioning data.  Hard reset returns application to factory default settings
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0 = None	
			Valid value set: 0=None 1 = Stop 2 = Start 3 = Soft Reset 4= Hard Reset	

<b>Standard object type name: UAP management object</b>				
<b>Standard object type identifier: 1</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
MaxRetries	4	The maximum number of client request retries this application process will send in order to have a successful client/server communication.	Type: Unsigned8	The number of retries sent for a particular message may vary by message based on application process determination of the importance of the message.
			Classification: Static	For example, some messages may not be retried at all, and others may be retried the maximum number of times
			Accessibility: Read only	
			Initial default value: 3	
			Valid value set: 0 to 8	
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: As valid for communication contract data type	
Number of objects in the UAP including this UAPMO	8	Number of objects in the UAP including this UAPMO	Type: Unsigned8	All UAPs are required to have a UAPMO, hence the default value is indicated as being 1 (one). The actual value of this attribute shall be the total number of objects contained in the UAP, including the UAPMO
			Classification: Static	
			Accessibility: Read only	
			Initial default value: 1	
			Valid value set: 1..255	
Array of UAP contained objects	9	Identification of the objects and type contained in this UAP	Type: Array of ObjectIDandType	See Table 271
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set: As valid for corresponding data type	
Static_Revision_Level	10	Revision level of the static data associated with all management objects	Type: Uint16	Revision level is incremented each time a static attribute value of any object contained in this UAP is changed
			Classification: Static	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set: 0 to 0xFFFF	
Reserved for future use by this standard	5 through 7, and 11 through 63			N octets of presently undefined content

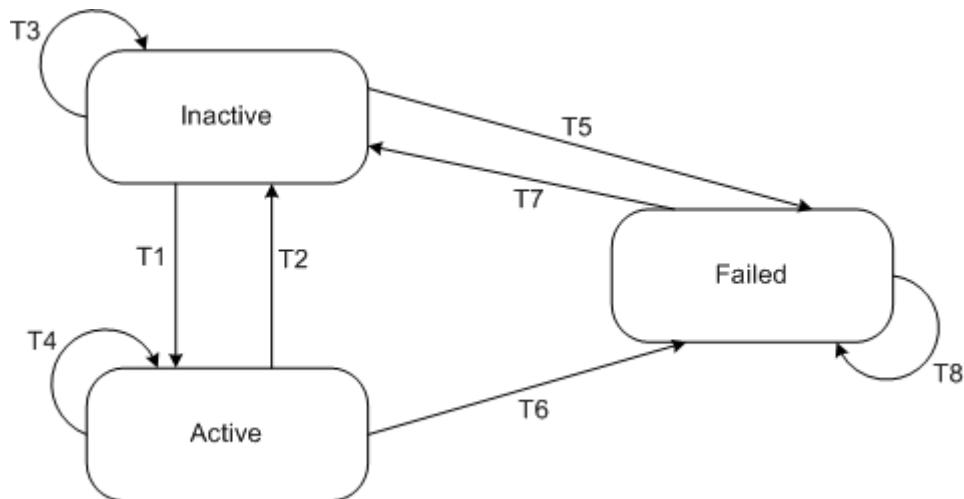
### 12.15.2.2.3 State table for user application process management object

Table 241 describes the state table for the UAP management object.

**Table 241 – State table for UAP management object**

<b>Transition</b>	<b>Current state</b>	<b>Event(s)</b>	<b>Action(s)</b>	<b>Next state</b>
T1	Inactive	Write(Command, Start)	Write.rsp(success)	Active
T2	Active	Write(Command, Stop)	Write.rsp(success)	Inactive
T3	Inactive	Write (Command,Stop)	Write.rsp(success)	Inactive
		Write(any Reset command)	Write.rsp(operationAccepted)	
T4	Active	Write(Command,Start)	Write.rsp(success)	Active
		Write(any other command)	Write.rsp(objectStateConflict)	
T5	Inactive	Write(Command,Start)	Write.rsp(failed) Note: Fails to start	Failed
T6	Active	Application problem	N/A	Failed
T7	Failed	Write(Any Reset command)	Write.rsp(operationAccepted)	Inactive
T8	Failed	Write(Any command other than Reset)	Write.rsp(objectStateConflict)	Failed

Figure 120 shows the UAP management object state diagram.

**Figure 120 – UAP management object state diagram****12.15.2.2.4 Standard object methods**

A UAP management object has methods as defined in Table 242.

**Table 242 – UAP management object methods**

<b>Standard object type name:</b> UAP Management Object		
<b>Standard object type identifier:</b> 1		
<b>Defining organization:</b> ISA		
Method name	Method ID	Method description
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0 through 127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128 through 255	These method identifiers are available for implementation-specific use

**12.15.2.3 Alert receiving object****12.15.2.3.1 General**

There may be up to four alert receiving objects in a device, one per alert reporting category. These alert receiving objects may receive more than one category of alert report. Categories of alert reports received by alert objects shall be unique; that is, if one alert receiving object is receiving alerts of category X from the ASL, no other alert objects in the device may also receive alerts of category X from the ASL. These alert receiving objects may be contained in the same or different processes (e.g., an alert receiving object for security alerts may be contained in a one application process, while another for process alerts may be contained in another application process).

NOTE Further separation of alerts, or consolidation and re-reporting of alerts, if necessary, is an application process local matter, outside the scope of the AL specification.

**12.15.2.3.2 Object attributes**

An alert receiving object may receive alerts from one or more alert reporting sources. The object has the attributes defined in Table 243.

**Table 243 – Alert receiving object attributes**

<b>Standard object type name: Alert receiving object</b>				
<b>Standard object type identifier: 2</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16	N/A
			Classification: Constant	
			Accessibility: N/A	
			Initial default value: N/A	
			Valid value set: 1 to 65535	
Reserved for future use	0			
Categories	1	Bitstring of alert categories indicating which object instance supports receiving	Type: Bitstring	N/A
			Classification: Static	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set: Bit 0 set True for device alerts; Bit 1 set True for communication alerts; Bit 2 set True for security alerts; Bit 3 set True for process alerts Bits 4 through 7 are reserved for future use by this standard	
Errors	2	Count of reports received not for a category that the receiving object indicated was supported	Type: Unsigned16	Wraps to 0 when maximum value is reached
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set: 0 to 65,536	
Reserved for future use by this standard	3-63			N octets of presently undefined content

**12.15.2.3.3 State table for AlertReport handling**

Table 244 indicates the states for handling reception of an AlertReport.

**Table 244 – State table for handling an AlertReport reception**

Transition	Current State	Event(s)	Action(s)	Next State
T1	Ready	AlertReport.ind received	Note processing alert report from Device X	Handle individual alert in report
T2	Handle individual alert in report	Check category	Valid category: Acknowledge alert report, and process it	Ready
			Invalid category: Increment the Alert Receiving Object instances value of its Errors attribute	

Figure 121 shows the state diagram for alert reception.

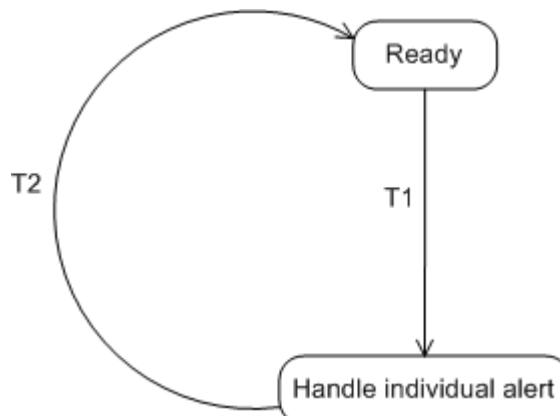
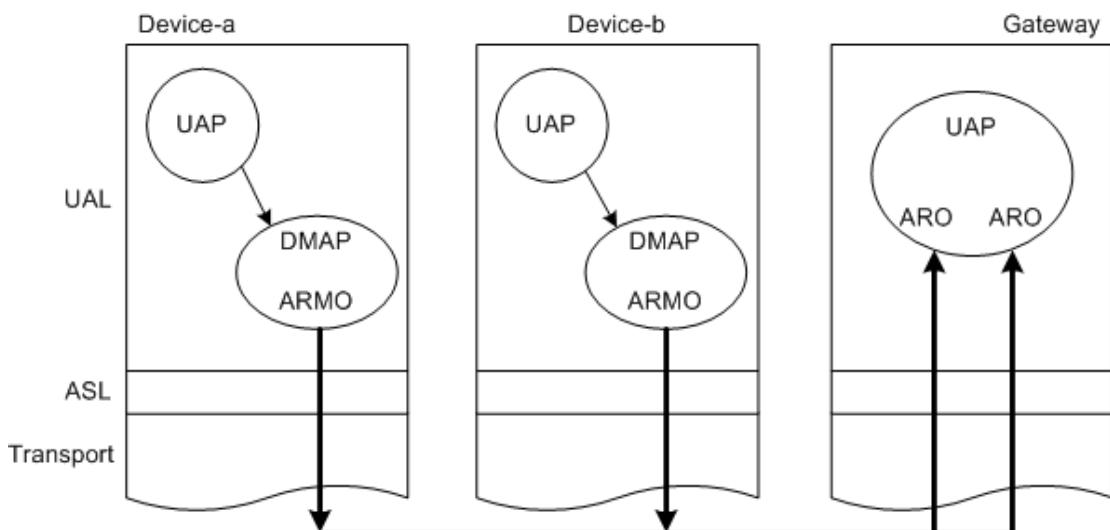
**Figure 121 – Alert report reception state diagram**

Figure 122 shows one example of alert reporting from multiple devices sources to multiple alert receiving objects contained in a single UAP of a single sink device.

**Figure 122 – Alert reporting example**

#### 12.15.2.3.4 Standard object methods

An AlertReceiving object has the methods defined in Table 245.

**Table 245 – AlertReceiving object methods**

Standard object type name: AlertReceiving object		
Standard object type identifier: 2		
Defining organization: ISA		
Method name	Method ID	Method description
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0-127	These method identifiers are reserved for future use by this standard.
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

#### 12.15.2.4 UploadDownload object

##### 12.15.2.4.1 General

An UploadDownload object is used for either uploading or downloading information to a device. The UploadDownload object may be used to support operations such as downloading a new version of operating firmware or downloading new UAP-contained code or UAP-required bulk data. The UploadDownload object maintains revision control information to indicate what was downloaded or what is available for upload (or both).

An UploadDownload object is likely to support upload or download for a single semantic set of information. An UploadDownload object shall support only one upload or download operation at a time.

A process may have zero or more UploadDownload object instances. Multiple UploadDownload object instances are required if more than one semantic set of information is needed to upload or download its required content.

NOTE 1 The local effect of an application process upload or download (e.g., a download may create new network-visible objects) is a local matter, outside the scope of the standard.

NOTE 2 An UploadDownload object may be used to support upload operations such as the upload of statistical or historical information from the device for analysis. An UploadDownload object may be used to update software/firmware in the target device.

NOTE 3 Support of multicast download may be considered in future revisions of this standard. To support multicast loads to a specific set of devices, a configuration tool is currently envisioned to be used to configure the multicast address/device/object relationships for the objects in the multicast set.

#### **12.15.2.4.2 Object attributes**

An UploadDownload object has the attributes defined in Table 246. Attributes are included in this object type in order to provide application-level communication timing guidance to the client that is communicating with the UploadDownload object.

NOTE Further guidance to the client, such as regarding tuning of communication timing (for example, related to network communication delays due to the topology of the messaging graph traversed, potential queuing delays, etc.), that may be used to tune client application behavior is transparent to an application process, and hence the application itself cannot provide complete guidance.

**Table 246 – UploadDownload object attributes**

<b>Standard object type name: UploadDownload object</b>				
<b>Standard object type identifier: 3</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Accessibility: N/A Initial default value: N/A Valid value set: 1 to 65535	N/A
Reserved for future use	0			
OperationsSupported	1	Indicates if this object supports uploads, downloads, or both	Type: Unsigned8 Classification: Constant Accessibility: Read only Initial default value: N/A  Valid value set: 0 = Defined size unicast upload only 1 = Defined size unicast download only 2 = Defined Size unicast upload and unicast download 3 - 15 reserved for future use by this standard	N/A
Description	2	Human readable identification of associated content.	Type: VisibleString Max length: 64 octets Classification: Static Accessibility: Read only Initial default value: N/A Valid value set: N/A	
State	3	State of the UploadDownload Object instance	Type: Unsigned8 Classification: Dynamic Accessibility: Read only Initial default value: 0 = Idle  Valid value set: 0=Idle 1=Downloading 2=Uploading 3=Applying 4=DLComplete 5=ULComplete 6=DLError 7=ULError	See state table below
Command	4	Action command to this object	Type: Unsigned8 Classification: Non-bufferable Accessibility: Read/write Initial default value: N/A  Valid value set: 0=Reset 1=Apply (used for Download only) 2 - 15 reserved for future use by this standard	See Table 254 below
MaxBlockSize	5	Maximum size of a block which can be accepted for a download, or provided for an upload	Type: Unsigned16 Classification: Static Accessibility: Read only  Initial default value: 1 to (MaxNPDUsize - Max TL header size - max(sizeof	The value, in units of octets, shall not exceed the maximum amount of data that can be conveyed in a single APDU per the communication contract.

<b>Standard object type name: UploadDownload object</b>				
<b>Standard object type identifier: 3</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
			(additional coding of Application layer UploadData service request), additional coding of sizeof(Application layer DownloadData service response))  Valid value set: 0...maximum size for data in an APDU	Additionally, space in the APDU shall be left for service related encoding.  Block sizes conveyed may be smaller than this value, but shall not be larger
MaxDownloadSize	6	Maximum size available for download as a whole.	Type: Unsigned32  Classification: Static  Accessibility: Read only  Initial default value: 0  Valid value set: 0 to 4,294,967,295	In octets
MaxUploadSize	7	Size available for Upload	Type: Unsigned32  Classification: Static  Accessibility: Read only  Initial default value: 0  Valid value set: 0 to 4,294,967,295	In octets
DownloadPrepTime	8	Time required, in seconds, to prepare for a download	Type: Unsigned16  Classification: Static  Accessibility: Read only  Initial default value: 0  Valid value set: 0 to 65535	Time required between sending the StartDownload response till the object can handle a DownloadData
DownloadActivationTime	9	Time in seconds for the object to apply newly downloaded content	Type: Unsigned16  Classification: Static  Accessibility: Read only  Initial default value: 0  Valid value set: 0 to 65535	N/A
UploadPrepTime	10	Time required, in seconds, to prepare for an upload	Type: Unsigned16  Classification: Static  Accessibility: Read only  Initial default value: 0  Valid value set: 0 to 65535	Time from sending the StartUpload response till the object can accept an UploadData
UploadProcessingTime	11	Typical time in seconds for this application object to process a request to upload a block	Type: Unsigned16  Classification: Static  Accessibility: Read only  Initial default value: 0  Valid value set: 0 to 65535	This information is intended to allow a client of an Upload operation to tune its upload related messaging to correspond to the operation of the particular UploadDownload object instance.  For example, a client may use this time to help determine its timeout/retry policy, or to determine when to invoke a method on the object instance
DownloadProcessingTime	12	Typical in seconds for this application object to process a downloaded block	Type: Unsigned16  Classification: Static  Accessibility: Read only  Initial default value: 0  Valid value set: 0 to 65535	This information may be used by a client of a Download operation to tune its download related messaging to correspond to the operation of the particular UploadDownload object instance.  For example, a client may use this time to help

<b>Standard object type name: UploadDownload object</b>				
<b>Standard object type identifier: 3</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
				determine its timeout/retry policy, or to determine when to invoke a method on the object instance
CutoverTime	13	Time (in seconds) specified to apply the download content	Type: TAINetworkTimeValue  Classification: Static  Accessibility: Read Write  Initial default value : 0  Valid value set: See definition of TAINetworkTimeValue	Downloaded content will be applied at the time specified by this attribute
LastBlockDownloaded	14	Number of last block successfully downloaded	Type: Unsigned16  Classification: Static  Accessibility: Read only  Initial default value : 0  Valid value set: 0 to 65535	Updated when an execute response to a DownloadData method is returned. Block number counting shall start at 1 (one). See 12.15.2.4.5.3.1
LastBlockUploaded	15	Number of last block successfully uploaded	Type: Unsigned16  Classification: Static  Accessibility: Read only  Initial default value : 0  Valid value set: 0 to 65535	Updated when an execute response to an UploadData method is returned. Block number counting shall start at 1 (one). See 12.15.2.4.5.3.1
ErrorCode	16	Upload or Download Error	Type: Unsigned8  Classification: Static  Accessibility: Read only  Initial default value : 0 =(noError)  Valid Values: Lower octet: 0 = noError 1 = timeout 2 = clientAbort 3 through 63 are reserved for future use by this standard,except: 18 = InconsistentContent 27 = InsufficientDeviceResources 64 through 255 = manufacturerSpecific	Updated when there is an error in uploading or downloading to this object. The error is cleared when the object transitions out of the error state to the Idle state. Use InconsistentContent to indicate that the device did not cutover as scheduled due to problem with download payload. Use InsufficientDeviceResources to indicate that the download could not be completed due to lack of memory or other resources
Reserved for future use by this standard	16-63			
NOTE 1 The standard does not prescribe product lifecycle management or versioning policies.				
NOTE 2 Description may be used to indicate interchangeability of versions, or to identify features / fixes / software builds.				
NOTE 3 The maximum value of the additional coding for the application coding for this standard is 9 octets.				
NOTE 4 Implementers may wish to consult IETF RFC 2348 regarding recommendations for a maximum size of communications.				

### 12.15.2.4.3 Standard object methods

Initiation of an upload or download bulk data transfer requires first reaching an agreement between the corresponding application objects to participate in the data transfer.

Any additional coordination required to ensure the readiness of the responding device to accept an upload or download request is the responsibility of the UAL process that will be starting a bulk transfer operation.

Client/server messaging to access coordination information from an attribute or set of attributes of the UploadDownload object may be used to support this coordination activity. Specifically, a read request may be used in advance of starting a bulk transfer in order for a

client to collect bulk transfer communication-related information specific to an UploadDownload object instance. Once agreement is reached, the client application controls when the data is provided (for a download) to the UploadDownload object, or requested (for an upload) from the UploadDownload object. When transfer is complete, the client indicates the transfer has ended. Additionally, the client may close the transfer if it determines that the entire data transfer should not be completed.

The serving application may request the data transfer be aborted if it determines that the data transfer cannot or should not be completed.

**NOTE 1** A serving application making a decision to abort based on lack of communication from the client should at least allow for the default standard retries and retry timing policy for the client/server communication policy in order to establish an appropriate timeout.

As with other application communications, it is required that transmission bandwidth be allocated by a communication service contract in order to support a bulk data transfer. Bandwidth for bulk data transfer is not considered dedicated bandwidth as used for periodic messaging, but rather is considered shared bandwidth as used for aperiodic messaging. Use of shared bandwidth among all users of shared bandwidth by a device is dependent on a combination of overall contract priority and message priority. Contract priority is defined by the system manager. Message priority is defined by the application process.

**NOTE 2** Any required coordination or sequencing of multiple images to different UploadDownload objects is the responsibility of the host application process. For each uploadable or downloadable image, a separate UploadDownload object instance needs to exist.

**NOTE 3** The semantics and syntax of the content and use of uploaded or downloaded information are outside the scope of this standard. The resulting activity in the application process of the device providing upload data or accepting download data, other than updating the UploadDownload object itself, is a local matter, and hence is outside the scope of this standard.

**NOTE 4** If a single device needs to process the download multiple times, a proxy application may be employed within the device to distribute the information internally. Internal distribution is purely a local implementation matter, and thus is outside the scope of this standard.

Upload from or download to a single device uses a unicast protocol; that is, the upload or download content is sent from/to a single UploadDownload object within a single device.

**NOTE 5** File content and/or naming conventions, if applicable to an upload or download are outside the scope of this standard.

An UploadDownload object has the methods defined in Table 247.

**Table 247 – UploadDownload object methods**

<b>Standard object type name: UploadDownload object</b>		
<b>Standard object type identifier: 3</b>		
<b>Defining organization: ISA</b>		
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>
Null	0	Reserved by standard for future use
StartDownload	1	This method is used by a client to reach an agreement with an UploadDownload object to participate in a download for which the client will be providing the data, one block at a time
DownloadData	2	This method is used by a client to provide data to an UploadDownload object for an agreed download operation
EndDownload	3	This method is used by a client to terminate a download operation that either has completed successfully, or which the client wishes to abort
StartUpload	4	This method is used by a client to reach an agreement with an UploadDownload object to participate in an upload for which the client will be requesting the data, one block at a time
UploadData	5	This method is used by a client to request data from an UploadDownload object for an agreed upload operation
EndUpload	6	This method is used by a client to terminate an upload operation that either has completed successfully, or that the client wishes to abort
Reserved for future use by this standard	7-127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

**NOTE** The approach used for upload and download has roots in the experiences of multiple accepted standards, including but not necessarily limited to Fieldbus Foundation, Device Net International, and IETF RFCs 1350, and

2347. Attributes of the UploadDownload object provide application-level information to assist in timeout interval determination by a client, hence IETF RFC 2349 is not followed. ACK retry as proposed in IETF RFC 2347 is not adopted for the following reasons:

- (1) The use cases driving this standard, and the agreed set of technical requirements this standard is to meet, have the vast bulk of communication being publish/subscribe, and very limited use of upload/download operations.
- (2) The upload/download operations that have been defined are not time-critical.
- (3) The client application receives feedback from the server if the server is getting duplicates, and can elect to terminate the operation.
- (4) The server application is aware of when it is sending error messages back to the client, and may elect to abort the operation.

#### **12.15.2.4.4 StartDownload method**

Table 248 describes the StartDownload method of the UploadDownload object.

**Table 248 – UploadDownload object StartDownload method**

<b>Standard object type name: UploadDownload object</b>			
<b>Standard object type identifier: 3</b>			
<b>Defining organization: ISA</b>			
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>	
StartDownload	1	A client uses the StartDownload method to indicate to an UploadDownload object instance that it desires to download the object.	
<b>Input arguments</b>			
<b>Argument #</b>	<b>Argument name</b>	<b>Argument type</b>	<b>Argument description</b>
1	BlockSize	Unsigned16	The size of a block of data in octets that will be downloaded.
2	DownloadSize	Unsigned32	The total size of data to be downloaded in octets.
3	DownloadMode	Unsigned8	The desired mode of operation.
<b>Output arguments</b>			
<b>Argument #</b>	<b>Argument name</b>	<b>Argument type</b>	<b>Argument description</b>
None			

#### **12.15.2.4.4.1 Method description**

A client uses the StartDownload method to indicate to an UploadDownload object instance that it desires to download the object, and to specify the parameters of the download in the input argument list. The UploadDownload object may accept or reject the download, indicating such via the output argument list.

If an UploadDownload object accepts to participate in a download operation, it shall not accept another download operation, or an upload operation until the download operation in process has been terminated or the object has been reset.

#### **12.15.2.4.4.2 Input arguments**

##### **12.15.2.4.4.2.1 BlockSize**

BlockSize indicates the size of a block of data in octets. All blocks shall have the same size, except that the last block of a download may contain a smaller positive number of octets.

##### **12.15.2.4.4.2.2 DownloadSize**

DownloadSize represents the total size of data to be downloaded, in octets.

##### **12.15.2.4.4.2.3 DownloadMode**

DownloadMode indicates the operational mode desired. The valid value for this argument indicates unicast and is represented by a value of zero.

#### **12.15.2.4.4.3 Output arguments**

There are no output arguments for this method.

#### 12.15.2.4.4.4 Response codes

The following feedback codes are valid for this method:

- operationAccepted;
- invalidBlockSize;
- invalidDownloadSize;
- unexpectedMethodSequence;
- insufficientDeviceResources;
- deviceHardwareCondition;
- vendorDefined; and
- other.

#### 12.15.2.4.5 DownloadData method

##### 12.15.2.4.5.1 General

Table 249 describes the DownloadData method of the UploadDownload object.

**Table 249 – UploadDownload object DownloadData method**

<b>Standard object type name: UploadDownload object</b>			
<b>Standard object type identifier: 3</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID	<b>Method description</b>	
DownloadData	2	A client uses the DownloadData method to provide data to an UploadDownload object that has agreed to be downloaded.	
<b>Input arguments</b>			
Argument #	Argument name	Argument type	Argument description
1	BlockNumber	Unsigned16	BlockNumber being downloaded
2	Data	Octet string (sequence of one or more octets)	The data for the block being downloaded. The maximum length of this string may vary, such as it may differ for different destination UploadDownload objects.
<b>Output arguments</b>			
Argument #	Argument name	Argument type	Argument description
1	CurrentBlockNumber	Unsigned16	This argument is present if the serviceFeedbackCode indicates either block out of sequence or duplicate.

##### 12.15.2.4.5.2 Method description

StartDownload is used first to have the UploadDownload object agree to the download.

Data is sent one block at a time, sequentially from the lowest numbered block to the highest numbered block using the DownloadData method. Only one DownloadData method invocation may be outstanding at a time; for example, if DownloadData for block n has been invoked, DownloadData for block n+1 shall not be invoked until a successful response containing the output arguments for the download of block n has been received by the client.

The UploadDownload object may indicate that it needs to abort via output argument MethodStatus.

NOTE 1 If a client of an upload or download operation is issuing multiple data transfer method invocations for the same block, it may be due to either a network-related problem (e.g., the request is not reaching the server) or a problem at the server device. In this situation, the client may employ the appropriate operation end method (EndDownload or EndUpload) to terminate the operation.

NOTE 2 If a client of an upload or download receives multiple dataSequenceError responses, it may be due either to network related problems (for example, loss of a method invocation response), or problems at the server device. In this situation, the client may employ the appropriate operation end method (EndDownload or EndUpload) to terminate the operation. Correspondingly, if an UploadDownload object has sent multiple dataSequenceError responses, it may infer that there are either network-related problems or problems at the client device and may

elect to abort the operation. If an UploadDownload object indicates operation abort, and this abort is lost over the network, the response sent to a subsequent data method (DownloadData or UploadData) or end method (EndDownload or EndUpload) indicates that the object is no longer participating in an upload or download operation with this client by sending a response indicating unexpectedMethodSequence.

#### **12.15.2.4.5.3 Input arguments**

##### **12.15.2.4.5.3.1 BlockNumber**

BlockNumber is the number of the block for which data is provided. Block number counting shall start at 1 (one).

##### **12.15.2.4.5.3.2 Data**

Data represents the data for the block being downloaded.

#### **12.15.2.4.5.4 Output arguments**

This current BlockNumber argument is present if the serviceFeedbackCode indicates either block out of sequence or duplicate. The argument indicates the last BlockNumber received. The intent is to permit the client to resolve an out of sequence or duplicate block reception error without aborting the download operation.

#### **12.15.2.4.5.5 Response codes**

The following feedback codes are valid for this method:

- success;
- invalidBlockNumber;
- blockDataError (e.g., wrong block size; content problem);
- unexpectedMethodSequence;
- insufficientDeviceResources;
- deviceHardwareCondition;
- operationAborted;
- other;
- dataSequenceError (e.g., duplicate);
- timingViolation; and
- vendorDefined.

#### **12.15.2.4.6 EndDownload method**

##### **12.15.2.4.6.1 General**

Table 250 describes the EndDownload method of the UploadDownload object.

**Table 250 – UploadDownload object EndDownload method**

<b>Standard object type name:</b> UploadDownload object				
<b>Standard object type identifier:</b> 3				
<b>Defining organization:</b> ISA				
Method name	Method ID (non-negative)	<b>Method description</b>		
EndDownload	3	A client uses the EndDownload method to indicate that the download is terminating.		
<b>Input arguments</b>				
Argument #	Argument name	Argument type	Argument description	
1	Rationale	Unsigned8	This argument indicates the clients reason for terminating the download operation	
<b>Output arguments</b>				
Argument #	Argument name	Argument type	Argument description	
None				

#### **12.15.2.4.6.2 Method description**

A client uses the EndDownload method to indicate that the download operation is terminating. Termination may occur, for example, if the download has completed, or if the client has elected to terminate the download operation.

EndDownload may be sent from a client that is presently engaged in a download operation, as agreed by the StartDownload method.

#### **12.15.2.4.6.3 Input arguments**

The Rationale argument indicates the client's reason for terminating the download operation. The value used shall be from the following set:

- Download successfully complete, indicated by a value equal to 0; or
- Client abort, indicated by a value equal to 1.

#### **12.15.2.4.6.4 Output arguments**

There are no output arguments for this method.

#### **12.15.2.4.6.5 Response codes**

The following feedback codes are valid for this method:

- success;
- operationIncomplete;
- unexpectedMethodSequence;
- timingViolation;
- vendorDefined; and
- other.

#### **12.15.2.4.7 StartUpload method**

##### **12.15.2.4.7.1 General**

Table 251 describes the StartUpload method of the UploadDownload object.

**Table 251 – UploadDownload object StartUpload method**

<b>Standard object type name:</b> UploadDownload object			
<b>Standard object type identifier:</b> 3			
<b>Defining organization:</b> ISA			
Method name	Method ID	<b>Method description</b>	
StartUpload	4	A client uses the StartUpload method to indicate to a UploadDownload object instance that it desires to upload data from the object.	
<b>Input arguments</b>			
Argument #	Argument name	Argument type	Argument description
1	DownloadMode	Unsigned8	The desired mode of operation
<b>Output arguments</b>			
Argument #	Argument name	Argument type	Argument description
1	BlockSize	Unsigned16	The size of a block of data in octets
2	UploadSize	Unsigned32	The total size of the data to be uploaded in octets

#### **12.15.2.4.7.2 Method description**

A client uses the StartUpload method to indicate to an UploadDownload object instance that it desires to upload data from the object. The UploadDownload object may accept or reject the upload, indicating such via the output argument list.

If an UploadDownload object accepts to participate in an upload operation, it shall not accept another upload operation or a download operation until the upload operation in process has been terminated or the object has been reset.

#### **12.15.2.4.7.3 Input arguments**

##### **12.15.2.4.7.3.1 DownloadMode**

The DownloadMode argument specifies the desired mode of operation. The valid value for this argument indicates unicast and is represented by a value of zero.

#### **12.15.2.4.7.4 Output arguments**

##### **12.15.2.4.7.4.1 BlockSize**

BlockSize is the size of a block of data in octets. All blocks shall have the same size, except that the last block of an upload may contain a smaller positive number of octets.

##### **12.15.2.4.7.4.2 UploadSize**

This argument indicates the size of the data to be uploaded, in octets.

#### **12.15.2.4.7.5 Response codes**

The following feedback codes are valid for this method:

- success;
- unexpectedMethodSequence;
- insufficientDeviceResources;
- deviceHardwareCondition;
- vendorDefined; and
- other.

#### **12.15.2.4.8 UploadData method**

##### **12.15.2.4.8.1 General**

Table 252 describes the UploadData method of the UploadDownload object.

**Table 252 – UploadDownload object UploadData method**

<b>Standard object type name:</b> UploadDownload object			
<b>Standard object type identifier:</b> 3			
<b>Defining organization:</b> ISA			
Method name	Method ID	<b>Method description</b>	
UploadData	5	A client uses the UploadData method to acquire data from an UploadDownload object that has agreed to be uploaded.	
<b>Input arguments</b>			
Argument #	Argument name	Argument type	Argument description
1	BlockNumber	Unsigned16	The number of the block for which data is requested
<b>Output arguments</b>			
Argument #	Argument name	Argument type	Argument description
1	Data	Octet string	This argument contains the data for the requested block. This argument is present if and only if the serviceFeedbackCode indicates success. The maximum size of this may vary by UploadDownload object instance being uploaded

#### **12.15.2.4.8.2 Method description**

A client uses the UploadData method to acquire data from an UploadDownload object which has agreed to be uploaded.

The StartUpload is used first to have the UploadDownload object agree to the upload. Data is requested one block at a time, sequentially from the lowest numbered block to the highest numbered block. Only one UploadData method invocation may be outstanding at a time. For example, if UploadData for block n has been invoked, UploadData for block n+1 shall not be invoked until the corresponding successful response containing the output arguments has been received by the client.

The UploadDownload object may indicate that it needs to abort via an output argument.

#### **12.15.2.4.8.3 Input arguments**

The BlockNumber argument specifies the number of the block for which data is requested. Block number counting shall start at 1 (one).

#### **12.15.2.4.8.4 Output arguments**

The Data argument contains the data for the requested block. This argument is present if and only if the serviceFeedbackCode indicates success.

#### **12.15.2.4.8.5 Service feedback codes**

The following feedback codes are valid for this method:

- success;
- unexpectedMethodSequence;
- insufficientDeviceResources;
- deviceHardwareCondition;
- operationAborted;
- other;
- dataSequenceError (e.g., duplicate, invalid block number, unexpected block number);
- timingViolation; and
- vendorDefined.

### 12.15.2.4.9 EndUpload method

#### 12.15.2.4.9.1 General

Table 253 describes the EndUpload method of the UploadDownload object.

**Table 253 – UploadDownload object EndUpload method**

<b>Standard object type name: UploadDownload object</b>			
<b>Standard object type identifier: 3</b>			
<b>Defining organization: ISA</b>			
Method name	Method ID (non-negative)	<b>Method description</b>	
EndUpload	6	A client uses the EndUpload method to indicate that the upload operation is terminating.	
<b>Input arguments</b>			
Argument #	Argument name	Argument type	Argument description
1	Rationale	Unsigned8	This argument indicates the clients reason for terminating the upload operation
<b>Output arguments</b>			
Argument #	Argument name	Argument type	Argument description
None			

#### 12.15.2.4.9.2 Method description

A client uses the EndUpload method to indicate that the upload operation is terminating. Termination may occur for example if the upload has completed, or if the client has elected to terminate the upload operation.

EndUpload may be sent from a client that is presently engaged in an upload operation, as agreed by the StartUpload method.

#### 12.15.2.4.9.3 Input arguments

The Rationale argument indicates the client's reason for terminating the upload operation. The value used shall be from the following set:

- Upload successfully complete, indicated by a value of zero; or
- Client abort, indicated by a value of one.

#### 12.15.2.4.9.4 Output arguments

There are no output arguments for this method.

#### 12.15.2.4.9.5 Service feedback codes

The following feedback codes are valid for this method:

- success;
- operationIncomplete;
- unexpectedMethodSequence ;
- timingViolation;
- vendorDefined; and
- other.

#### 12.15.2.4.10 State table for download

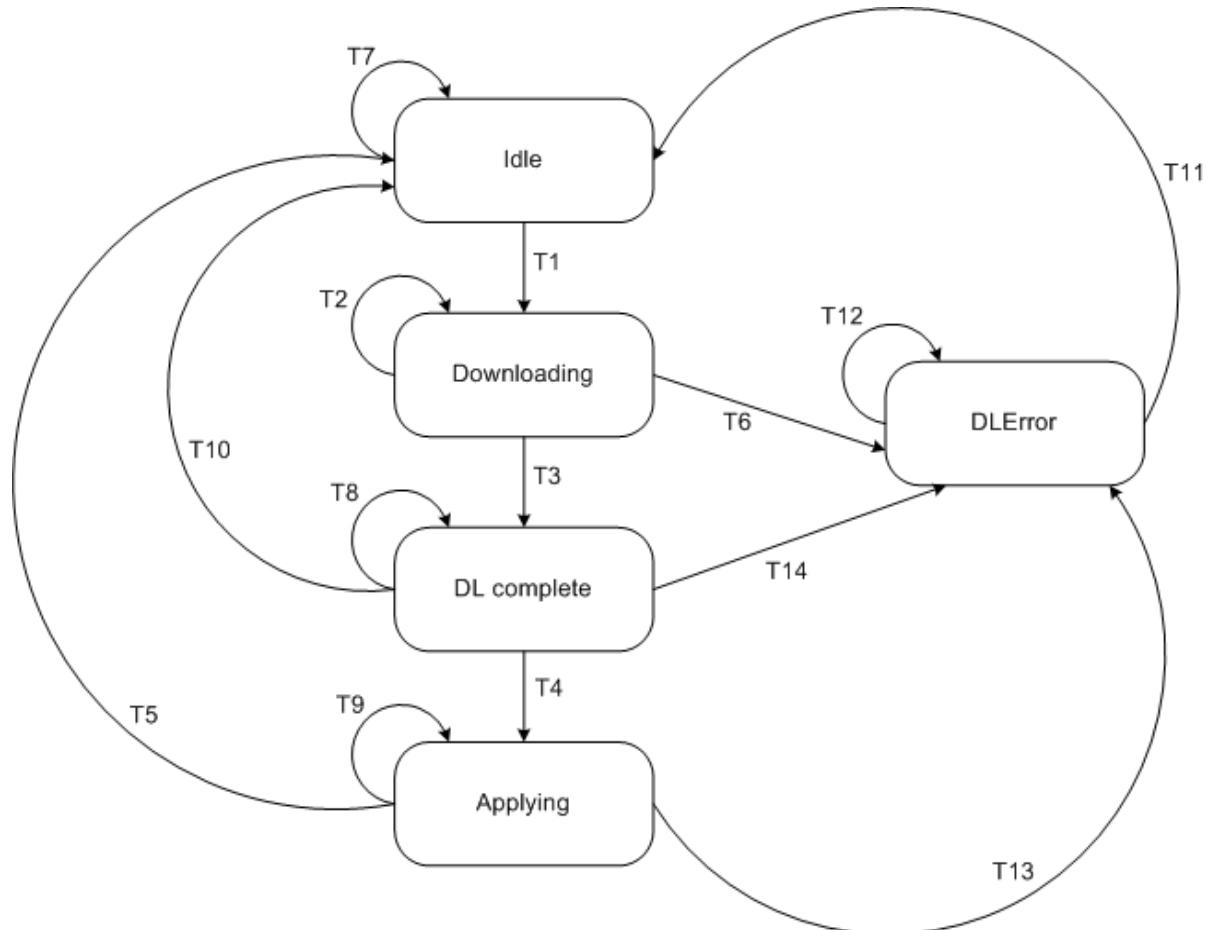
Table 254 shows the download state table.

**Table 254 – Download state table for unicast operation mode**

Transition	Current State	Event(s)	Action(s)	Next State
T1	Idle	Execute.indicate(StartDownload)	Execute.response(success)	Downloading
T2	Downloading	Execute.indicate(DownloadData)  Request for block is from same client object that started the download, and download data parameters are acceptable.	Execute.response(success)	Downloading
		Execute.indicate(StartDownload) or Execute.indicate(any Upload Method)	Execute.response(objectStateConflict)	
		Execute.indicate(DownloadData) and request is from wrong client, or something is wrong with the download data parameters or timing.	Execute.response(appropriate error)  Where the appropriate error may be, for example, invalidArgument, incompatibleMode, timingViolation, ...  NOTE It is a local matter for the UploadDownload object to determine if/when to abort the download.	
		Execute.indicate(EndDownload [Success]) and UploadDownload object does not agree download was completed successfully.	Execute.response(incompatibleMode)	
		Write.indicate(StateCommand.An y value)	Write.response(objectStateConflict)	
T3	Downloading	Execute.indicate(EndDownload [Success])	Execute.response(Success)	DLComplete
T4	DLComplete	Write.indicate(StateCommand, Apply)	Write.response(operationAccepted)	Applying
T5	Applying	Application successful	None	Idle
T6	Downloading	Timeout waiting for subsequent method invocation	Update “ErrorCode” attribute of UploadDownload object	DLError
		Execute.indicate(EndDownload[Abort])	Update “ErrorCode” attribute of UploadDownload object. Execute.response (Success)	
T7	Idle	Execute.indicate(Any Download method other than StartDownload)	Execute.response(objectStateConflict )	Idle
		Execute.indicate(StartDownload) and Request is unacceptable. For example, one or more input arguments are not agreeable.	Execute.response(appropriate error) (e.g., invalidObjectID)	
		Write.indicate(StateCommand.An y value other than Reset)	Write.response(objectStateConflict)	
		Write.indicate(StateCommand.Re set)	Write.response(success)	
T8	DLComplete	Execute.indicate(Any Download method or any Upload method)	Execute.response(objectStateConflict )	DL_Complete
T9	Applying	Execute.indicate(Any download method or any upload method)	Execute.response(objectStateConflict )	Applying
		Write.indicate(StateCommand.An y value)	Write.response(objectStateConflict)	
T10	DLComplete	Write(StateCommand, Reset)	1. Discard download content; and 2. Write.req(success)	Idle
T11	DLError	Write(StateCommand, Reset)	1. Discard download content; 2. Clear “ErrorCode” attribute; and 3. Write.req(success)	Idle
T12	DLError	Any upload or download method	Execute.response (objectStateConflict)	DLError
		Any state command other than Write(StateCommand.Reset)	Write.req(objectStateConflict)	

T13	Applying	Application failure	Update "ErrorCode" attribute of UploadDownload object	DLError
T14	DLComplete	Timeout waiting for command to apply	Update "ErrorCode" attribute of UploadDownload object	DLError

Figure 123 shows the upload/download object download state diagram.



**Figure 123 – Upload/Download object download state diagram**

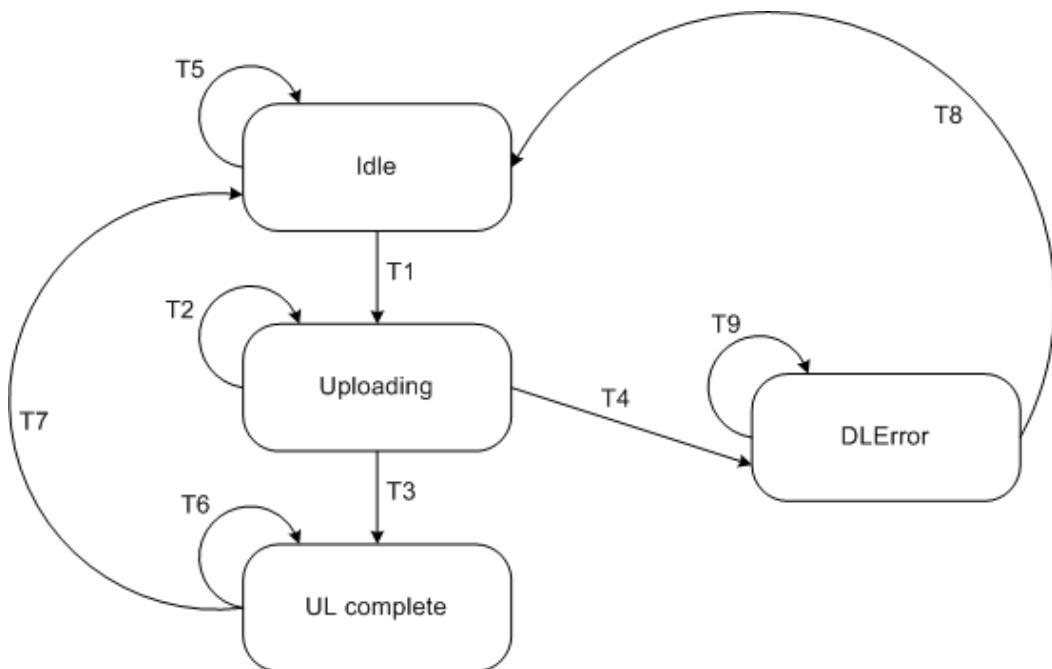
#### 12.15.2.4.11 State table for upload

Table 255 shows the upload state table.

**Table 255 – Upload state table for unicast operation mode**

Transition	Current State	Event(s)	Action(s)	Next State
T1	Idle	Execute.indicate(StartUpload)	Execute.response(Success)	Uploading
T2	Uploading	Execute.indicate(UploadData)  Request for block is from same client object that started the upload, and upload data parameters are as acceptable.	Execute.response(Success)	Uploading
		Execute.indicate (StartUpload) or any Download method	Execute.response (objectStateConflict)	
		Execute.indicate(UploadData) and request is from wrong client, or something is wrong with the upload data parameters or timing.	Execute.response(appropriate error)  Where the appropriate error may be, for example, invalidArgument, incompatibleMode, timingViolation, ...  NOTE It is a local matter for the UploadDownload object to determine if/when to abort the upload if this occurs more than once consecutively.	
		Execute.indicate(EndUpload [Success]) and UploadDownload object does not agree upload was successful	Execute.response(incompatibleMode)	
		Write.indicate(StateCommand .Any value)	Write.response(objectStateConflict)	
T3	Uploading	Execute.indicate(EndUpload [Success])	Execute.response(Success)	ULComplete
T4	Uploading	Timeout waiting for subsequent method invocation	Update “ErrorCode” attribute of UploadDownload object	UL_Error
		Execute.indicate(EndUpload [Abort])	1. Update “ErrorCode” attribute of UploadDownload object; 2. Execute.response (Success)	
T5	Idle	Execute.indicate(Any Upload method other than StartUpload);	Execute.response(objectStateConflict)	Idle
		Execute.indicate(StartUpload) and Request is unacceptable. For example, one or more input arguments are not agreeable.	Execute.response(appropriate error) (e.g., invalidObjectID)	
		Write.indicate(StateCommand. any other than Reset)	Write.response(objectStateConflict)	
		Write.indicate(StateCommand. Reset)	Write.response(success)	
T6	ULComplete	Execute.indicate(Any DownloadMethod) or Any upload method)	Execute.response (objectStateConflict)	UL_Complete
T7	ULComplete	Write(StateCommand, Reset)	Write.req(success)	Idle
T8	ULError	Write(StateCommand, Reset)	1. Clear “ErrorCode attribute; and 2. Write.req(success)	Idle
T9	ULError	Any upload or download method, or	Execute.response (objectStateConflict)	ULError
		Write(StateCommand.other than Reset)	Write.req (error)	

Figure 124 shows the upload/download object upload state diagram.



**Figure 124 – Upload/Download object upload state diagram**

#### 12.15.2.4.12 Client responsibilities for upload/download operations

In order to handle message delays in both requests and responses, and to avoid a congestion collapse due to a retransmission loop, only the first instance of a response indicating success shall cause the next data block to be sent via a DownloadData or requested via an UploadData method invocation by the client.

**NOTE** The intent is to avoid recreating historical situations such as occurred with trivial file transfer protocol (TFTP), creating the Sorcerer's Apprentice Syndrome.

#### 12.15.2.5 Concentrator object

##### 12.15.2.5.1 General

A concentrator object represents an assembly of data, collected from multiple objects in the same UAP, that is to be published by a single publish request service. This object optimizes publication messages sent from a device. Multiple concentrator object instances may be used to represent multiple assemblies of data if required. A list of attributes is provided to indicate the data values that are published.

**NOTE** The published content represented by this object is established by configuration. This standard does not specify the device configuration tool.

A subscriber to data produced by a concentrator object shall only be a dispersion object. The data types associated with the list of attributes of the dispersion object should be configured to match those produced by the concentrator object.

When a concentrator object is configured by a host application, such as a gateway, the device is responsible for establishing contracts as needed to support the corresponding publications. The design is intended to support two use cases. In one case, the device joins the network and then the host configures the concentrator object. In the other case, the concentrator object is pre-configured and the device autonomously starts publication after it joins the network.

A UAP may have zero or more concentrator object instances.

##### 12.15.2.5.2 Object attributes

A concentrator object has the attributes defined in Table 256.

The first time a UAP receives a read/write/execute request from an endpoint for which it has no contract, it shall request a contract so that it can send a service response to the requesting endpoint. The UAP shall, as necessary, delay the first service response to allow for time to establish/modify the contract.

**Table 256 – Concentrator object attributes**

<b>Standard object type name: Concentrator object</b>				
<b>Standard object type identifier: 4</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Accessibility: N/A Initial default value: N/A Valid value set: 1 to 65535	N/A
Reserved for future use	0			
ConcentratorContentRevision	1	Tracks a change in what is published; ensures Concentrator (publisher) and Dispersion (subscriber) objects are in harmony	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 0 to 255	Revision shall be incremented when the complement of data to publish changes, i.e. CommunicationEndpoint or Array of ObjectAttributeIndexAndSize are changed. Attribute included in Table 347 header.
CommunicationEndpoint	2	Serves to identify the object that receives the publication from this object	Type: Communication association endpoint structure Classification: Static Accessibility: Read/write Initial default value: The configured connection endpoint valid element indicates not configured (i.e., endpoint is not valid) Valid value set: See structure type definition	Write to this attribute last when configuring this object; see Table 265
Communication contract data for scheduled communication	3	Data corresponding to the communication contract	Type: Communication contract data Classification: Static Accessibility: Read only Initial default value: N/A Valid value set: See communication contract data type definition	Updated when the corresponding contract is established or terminated; see Table 266
MaximumItemsPublishable	4	Maximum number of items that can be published	Type: Unsigned8 Classification: Constant Accessibility: Read only Initial default value: Local matter Valid value set: 0 to 255	If this attribute has a value of 0, it indicates it is not configured for publishing
NumberItemsPublishing	5	Actual number of items being published	Type: Unsigned8 Classification: Static Accessibility: Read only Initial default value: 0 Valid value set: 0 to 255	Updated as ObjectAttributeIndexAndSize attributes are configured : incremented when another value to publish is added, and decremented when a value to publish is removed
Array of ObjectAttributeIndexAndSize	6	Array of data to identify each piece of data published	Type: Array of ObjectAttributeIndexAndSize Classification: Static Accessibility: Read/write Initial default value: Element size is 0 Valid value set: As valid for corresponding data type	Object ID, attribute ID, attribute index, and size for each value published. See Table 264
Reserved for future use by this standard	7-63			

**NOTE** It is recommended that the Revision, NumItemsPublishing, and ObjAttrIdx attributes be implemented to be written to atomically by concatenation.

### 12.15.2.5.3 Standard object methods

A concentrator object has the methods defined in Table 257.

**Table 257 – Concentrator object methods**

<b>Standard object type name: Concentrator object</b>		
<b>Standard object type identifier: 4</b>		
<b>Defining organization: ISA</b>		
Method name	Method ID	Method description
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0-127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

### 12.15.2.6 Dispersion object

#### 12.15.2.6.1 General

A dispersion object is the subscribing object corresponding to a concentrator object. This object is configured to indicate how to parse a concentrator object's published content. If multiple disassemblies are required, multiple dispersion user objects are to be used. A UAP may have zero or more dispersion object instances.

**NOTE** Concentrator and dispersion objects are special objects supporting publication proxy within a UAP. These objects should not be confused with proxy application processes, which may distribute information across multiple UAPs within the UAL.

#### 12.15.2.6.2 Object attributes

A dispersion object has the attributes defined in Table 258.

**Table 258 – Dispersion object attributes**

<b>Standard object type name: Dispersion object</b>				
<b>Standard object type identifier: 5</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16	N/A
			Classification: Constant	
			Accessibility: N/A	
			Initial default value: N/A	
			Valid value set: 1 to 65535	
Reserved for future use	0			
ConcentratorContentRevision	1	Tracks changes to content subscribed. Ensures Concentrator publishing object and Dispersion subscribing object are in harmony	Type: Unsigned8	Updated when the complement of data to publish changes. In the event of a mismatched ContentRevision (Table 347), the publication shall not be processed
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0	
			Valid value set: 0 to 255	
CommunicationEndpoint	2	Endpoint of concentrator object that publishes data to this dispersion object	Type: Communication association endpoint structure	Write to this attribute last when configuring this object
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: The configured connection endpoint valid element indicates not configured (i.e., endpoint is not valid)	
			Valid value set: See structure definition	
MaximumItemsSubscribing	3	Maximum number of items that can be subscribed	Type: Unsigned8	Maximum # of items in corresponding publication
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: Local matter	
			Valid value set: 1 to 255	
NumItemsSubscribing	4	Number of items being subscribed to	Type: Unsigned8	Actual # of items in corresponding publication
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: 0	
			Valid value set: 0 to 255	
Array of ObjectAttributeIndexAndSize	5	Array of data to identify each piece of data published	Type: Array of ObjectAttributeIndexAndSize	Object ID, Attribute ID, Attribute index, and size of data for the destination within the application for the published information.  NOTE: To skip over data, the destination object and attribute may locally represent a Null object and Null attribute.
			Classification: : Static	
			Accessibility: Read/write	
			Initial default value: Element size is 0	
			Valid value set: N/A	
Reserved for future use by this standard	6-63			

NOTE It is recommended that the Revision, NumItemsSubscribing, and ObjAttrIdx attributes be implemented to be written to atomically by concatenation.

### 12.15.2.6.3 Standard object methods

A dispersion object has the methods defined in Table 259.

**Table 259 – Dispersion object methods**

<b>Standard object type name: Dispersion object</b>		
<b>Standard object type identifier: 5</b>		
<b>Defining organization: ISA</b>		
Method name	Method ID	Method description
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0-127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

### 12.15.2.7 Tunnel object

#### 12.15.2.7.1 General

The tunnel object (TUN) is used to support the energy efficient transport of encapsulated messages over the network for a single non-native protocol. The tunnel service and a variation of the publication service are defined for this encapsulation. Support structures are provided for deconstruction, mapping and reconstruction of non-native protocol packets in order to reduce transactions and packet size.

NOTE The usage of the tunnel object to create protocol translators is intended to be defined by the organization that has defined the non-native protocol used in the tunnel.

#### 12.15.2.7.2 Object attributes

A tunnel object has the attributes defined in Table 260.

**Table 260 – Tunnel object attributes**

<b>Standard object type name: Tunnel object</b>				
<b>Standard object type identifier: 6</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Accessibility: N/A Initial default value: N/A Valid value set: 1 to 65535	N/A
Reserved for future use	0			
Protocol	1	Type of protocol supported by this object	Type: Unsigned8 Classification: Constant Accessibility: Read only Initial default value: Local matter (protocol-specific) Valid value set: Defined in Annex M	Sets the specific protocol that is encapsulated in tunnel messages. Only matching protocol tunnels exchange meaningful data.
Status (Configuration status)	2	Communication configuration status of this object	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 0 -> not configured (tunnel configuration not valid) 1 -> configured (tunnel configuration valid) 2 -> configuration failed (tunnel configuration invalid)	The object status is not configured when the Protocol is set to None and no communication occurs. Once the object is configured and another protocol is set, the object attempts to apply the configuration and changes the status appropriately.
Flow_Type	3	Communication service utilized by this object	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: 0 -> 2 part tunnel 1 -> 4 part tunnel 2 -> publish 3 -> subscribe	Configures the tunnel for a specific type of communication and role.
Update_Policy	4	Periodic communication update policy for this object	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: 0 -> periodic 1 -> change of state	Sets the periodic publication policy for a linked publisher and subscriber. A periodic update publishes on every opportunity. Change of state publishes on fresh data or at least as often as Stale_Limit specifies.
Period (Data publication period)	5	Periodic communication update period for this object	Type: Integer16 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 1 to 65535	Sets the periodic publication time for a linked publisher and subscriber. Isochronous publication is enabled by an implicit rule. Publication does not begin until a period is set. Units in seconds. See 12.12.5
Phase (Ideal publication phase)	6	Periodic communication phase within period	Type: Unsigned8 Classification: Static Accessibility: Read/write	Sets the requested publication time within the period for a linked

<b>Standard object type name:</b> Tunnel object <b>Standard object type identifier:</b> 6 <b>Defining organization:</b> ISA				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
		for this object	Initial default value: N/A Valid value set: 0 to 99	publisher and subscriber. The actual phase may differ by contract requirements. Units should be indicated as a percentage %
Stale_Limit (Stale data limit)	7	Periodic communication stale data limit for this object	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: 0 to 255	Defines the maximum subscriber expected arrival time as a multiple of the period.  Defines the minimum publication rate for change of state reporting as a multiple of the period
Max_Peer_Tunnels	8	Maximum number of correspondent tunnels with which this object can communicate	Type: Unsigned8 Classification: Constant Accessibility: Read only Initial default value: 0 Valid value set: 0 to 255	N/A
Num_Peer_Tunnels	9	Actual number of correspondent tunnels with which this object is communicating	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 0 to 255	Incremented / decremented as Tunnel endpoints array elements are added and deleted
Array of Tunnel endpoint	10	Array of Protocol association endpoints	Type: Array of Tunnel endpoint Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: Address information pointing to a one or more tunnel objects	Links remote tunnel objects for communication with this tunnel object.
Foreign_Source_Address	11	Foreign source address mapped to this objects communication	Type: Unsigned128 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: Defined by protocol translator	Holds static addressing information to be delivered to initiator or correspondent upon message receipt.
Foreign_Destination_Address	12	Foreign destination address mapped to this objects communication	Type: Unsigned128 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: Defined by protocol translator	Holds static addressing information to be delivered to initiator or correspondent upon message receipt.
Connection_Info[ ]	13	Foreign connection information mapped to this objects communication	Type: OctetString Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: Defined by protocol translator	Holds static information to be delivered to initiator or correspondent upon message receipt.
Transaction_Info[ ]	14	Foreign transaction information mapped to this objects communication	Type: OctetString Classification: Dynamic Accessibility: Read/write Initial default value: N/A Valid value set: Defined by protocol translator	Holds transaction specific information to be delivered to initiator on completion of a transaction.

<b>Standard object type name: Tunnel object</b>				
<b>Standard object type identifier: 6</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Reserved for future use by this standard	15-63			

### 12.15.2.7.3 Standard object methods

A tunnel object has the methods defined in Table 261.

**Table 261 – Tunnel object methods**

<b>Standard object type name: Tunnel object</b>		
<b>Standard object type identifier: 6</b>		
<b>Defining organization: ISA</b>		
Method name	Method ID	Method description
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0-127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

### 12.15.2.8 Interface object

#### 12.15.2.8.1 General

The interface object provides a generic messaging end point for interfacing to a network. This object may be used as the source or destination object in native messaging interactions required for support of gateway protocol translation and various native messaging applications.

The interface object may be indicated in client/server communication services necessary for native read, write, and execute services. The interface object may also be referenced as the client object communicating with an upload/download object for bulk transfer.

Communications referencing the interface object as a client shall adhere to the client/server congestion control policies defined in this standard.

Where possible, implementers shall consider buffering client server retrieved values for local usage rather than creating additional communications over the wireless network to repeatedly retrieve these values from devices which need to preserve power. User application objects contained in the field devices provide guidance, via their specification of attribute data classification, regarding what object-related information should be buffered.

NOTE 1 Native object publication and subscription is accomplished by utilizing the concentrator and dispersion objects.

NOTE 2 The actual structure of buffering/cache and local requirements for handling messages to the cache are considered local matters, outside the scope of this standard.

#### 12.15.2.8.2 Object attributes

An interface object has the attributes defined in Table 262.

**Table 262 – Interface object attributes**

<b>Standard object type name: Interface object</b>				
<b>Standard object type identifier: 7</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16	N/A
			Classification: Constant	
			Accessibility: N/A	
			Initial default value: N/A	
			Valid value set: 1 to 65535	
Reserved for future use	0			
Reserved for future use by this standard	1-63			

### 12.15.2.8.3 Standard object methods

An interface object has the methods defined in Table 263.

**Table 263 – Interface object methods**

<b>Standard object type name: Interface object</b>		
<b>Standard object type identifier: 7</b>		
<b>Defining organization: ISA</b>		
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0-127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

## 12.16 Data types

### 12.16.1 Basic data types

The basic data types supported for attributes are:

- Binary values;
- 8-, 16-, and 32-bit signed integers;
- 8-, 16-, 32-, 64- and 128-bit unsigned integers;
- IEC 60559 (IEEE 754) 32-bit and 64-bit floating point values;
- Strings representing visible text, a block of octets, or a sequence of bit values (Bitstring); or

NOTE By direction of membership, explicit indication for unicode data is not required for transmission over this network. The reasoning is that text strings are primarily envisioned to be used on higher level networks in devices that are less constrained in terms of memory use and power consumption for transmission.

- Time: TAITimeDifference, TAINetworkTimeValue, TAITimeRounded.

### 12.16.2 Industry-independent standard data structures

#### 12.16.2.1 General

Standard data structures used shall be the data structures conveyed by the protocol defined by this standard. Industry-independent standard data structures are summarized in Annex L of this standard.

NOTE Vendor-specific data structure definitions are not supported.

#### 12.16.2.2 Object, attribute, index, and size

The elements of ObjectAttributeIndexAndSize are shown in Table 264.

**Table 264 – Data type: ObjectAttributeIndexAndSize**

<b>Standard data type name: ObjectAttributeIndexAndSize</b>		
<b>Standard data type code: 469</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
ObjectID	1	Type: Unsigned16 Classification: Static Accessibility: Varies by use Valid value set: 0 to 65535
AttributeID	2	Type: Unsigned16 Classification: Static Accessibility: Varies by use Valid value set: 0 to 65535
AttributeIndex	3	Type: Unsigned16 Classification: Static Accessibility: Varies by use Valid value set: 0 to 65535
Size	4	Type: Unsigned16 Classification: Static Accessibility: Varies by use Valid value set: 0 to 65535 NOTE In practice, this maximum size depends on the capabilities of the device.

#### **12.16.2.3 Communication association endpoint**

The data structure shown in Table 265 is used for communication endpoints for both inputs and outputs.

**Table 265 – Data type: Communication association endpoint**

<b>Standard data type name: Communication association endpoint</b>		
<b>Standard data type code: 468</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
Network address of remote endpoint	1	<p>Type: Unsigned128 This is a logical construct configured for the device by the system manager.</p> <p>NOTE The system manager ensures that such configuration supports both device replacement and mobile device scenarios.</p> <p>Size: 16 octets (128 bits). Classification : Static Accessibility: Read/write Valid value set: Defined by IPv6</p>
Transport layer port at remote endpoint	2	<p>Type: Unsigned16 Size: 16 bits Classification : Static Accessibility: Read/write Valid value set : Defined by transport layer</p>
Object ID at remote endpoint	3	<p>Type: Unsigned16 Size: 16 bits Classification : Static Accessibility: Read/write Valid value set: 0 to 65535</p>
Stale data limit	4	<p>Type: Unsigned8 Size: 1 octet Classification : Static Accessibility: Read/write Valid value set: 0 to 255</p> <p>NOTE 1 This attribute is primarily of interest to a subscriber.</p> <p>NOTE 2 Count of consecutive stale input values a subscriber receives before the subscriber should consider the value previously received to Bad (Table 299). Staleness is indicated by an unchanging freshness sequence number (Table 347).</p>
Data publication period	5	<p>Type: Integer16 Size: 2 octets Classification : Static Accessibility: Read/write</p> <p>NOTE For units of time, see 12.12.5</p>
Ideal publication phase	6	<p>Type: Unsigned8 (as a percentage %) Size: 2 octets Classification : Static Accessibility: Read/write Valid value set: 0 to 99</p> <p>NOTE This attribute is primarily of interest to a publisher.</p>
PublishAutoRetransmit	7	<p>Type: Boolean Size: 1 octet Classification: Static Accessibility: Read/write Valid value set: 0: Transmit only if application content changed since last publication 1 : Transmit at every periodic opportunity (regardless of whether application content changed since last transmission or not)</p>
Configuration status	8	<p>Unsigned8 Size: 1 octet Classification: Static Accessibility: Read access Valid value set: 0 : not configured (connection endpoint not valid) 1: configured (connection endpoint valid)</p> <p>NOTE The data owner sets this element to a value of 0 to indicate</p>

		that the endpoint is not configured, and to 1 to indicate the endpoint is configured. An endpoint is considered not configured if the value of Object ID at remote endpoint is 0.
--	--	---

#### 12.16.2.4 Communication contract data

The data structure shown in Table 266 is used for the dynamic data important to the local application process that is associated with a particular communication contract.

NOTE 1 It is a local matter to ensure that applications are well-behaved in terms of the communication contracts they employ. The AL does not standardize the policing of compliance with requested contracts.

NOTE 2 As part of contract negotiation, sufficient information is provided to the contract requesting device in order to enable it to determine the maximum size APDU that may be sent to the correspondent of the contract. For example, if contract negotiation determines the maximum network service data unit (NSDU) size, then the type of security in effect for the contract can be locally determined for the contract. If the type of security in use is known, the transport header size can be locally acquired and subtracted from the maximum NPDU size, thus yielding the value for the maximum APDU size that may be used for communications employing the particular communication contract.

**Table 266 – Data type: Communication contract data**

<b>Standard data type name: Communication contract data</b>		
<b>Standard data type code: 470</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element type</b>
ContractID	1	Type: Unsigned16 Classification: Static Accessibility: Read only Default value : N/A Valid value set: The set of valid values is defined by system management
Contract_Status	2	Type: Unsigned8 : Enumeration representing: Classification: Static Accessibility: Read only Default value = 0 (endpoint_not_configured) Valid value set: 0 = endpoint_not_configured 1 = awaiting_contract_establishment 2 = contract_active_as_requested 3 = contract_active_negotiated_down 4 = awaiting_contract_termination 5 = contract_establishment_failed 6 = contract_inactive
Actual_Phase	3	Type: Unsigned8 (in units of percentage %) Size: 2 octets Classification: Dynamic Accessibility: Read only Default value: 0 (indicating not assigned) Valid value set: 0 to 99

NOTE Further information on the actual contract, such as the negotiated-down parameters, may be available from the DMAP and does not need to be maintained by the UAP.

#### 12.16.2.5 Alert communication endpoint

The data structure shown in Table 267 is used for communication endpoints for alert reports.

**Table 267 – Data type: Alert communication endpoint**

<b>Standard data type name:</b> Alert communication endpoint		
<b>Standard data type code:</b> 471		
Element name	Element identifier	Element scalar type
Network address of remote endpoint	1	Type: Unsigned128 This is a logical construct configured for the device by the system manager. NOTE The system manager ensures that such configuration supports both device replacement and mobile device scenarios. Size: 16 octets (128 bits). Classification : Static Accessibility: Read/write Valid value set: Defined by IPv6
Transport layer port at remote endpoint	2	Type: Unsigned16 NOTE 16 bits Classification : Static Accessibility: Read/write Valid value set : Defined by transport layer
Object ID at remote endpoint	3	Type: Unsigned16 Size: 16 bits Classification : Static Accessibility: Read/write Valid value set: 0 to 65535

**12.16.2.6 Tunnel endpoint**

The data structure shown in Table 268 is used in tunnel objects to identify remote tunnel endpoints for exchange of encapsulated payloads.

**Table 268 – Data type: Tunnel endpoint**

<b>Standard data type name:</b> Tunnel endpoint		
<b>Standard data type code:</b> 475		
Element name	Element identifier	Element scalar type
Network_Address (Network address of remote endpoint)	1	Type: Unsigned128 This is a logical construct configured for the device by the system manager.  NOTE The system manager ensures that such configuration supports both device replacement and mobile device scenarios.  Size: 16 octets (128 bits) Classification: Static Accessibility: Read/write Valid value set: Defined by IPv6
Transport_Port (Transport layer port at remote endpoint)	2	Type: Unsigned16 Size: 16 bits Classification: Static Accessibility: Read/write Valid value set : Defined by transport layer
OID (Object ID at remote endpoint)	3	Type: Unsigned16 Size: 16 bits Classification: Static Accessibility: Read/write Valid value set: 0 to 65535

**12.16.2.7 Alert report descriptor**

Elements of the alert report descriptor are shown in Table 269.

**Table 269 – Data type: Alert report descriptor**

<b>Standard data type name:</b> Alert report descriptor		
<b>Standard data type code:</b> 499		
Element name	Element identifier	Element type
Alert report disabled	1	Type: Boolean Size: 1 octet Classification: Static Accessibility: Read/write Default value : Local matter Valid value set: True = Disabled False = Enabled
Alert report priority	2	Type: Unsigned8 Size: 1 octet Classification: Static Accessibility: Read/write Default value : 0 Valid value set: 0 to 15, where 0 indicates the lowest priority and 15 indicates the highest priority

**12.16.2.8 Analog alarm descriptor**

The analog alarm descriptor is used to define alarm reporting for an analog value with a single reference condition. Its elements are shown in Table 270.

**Table 270 – Data type: Process control alarm report descriptor for analog with single reference condition**

<b>Standard data type name:</b> Process control alarm report descriptor for analog with single reference condition		
<b>Standard data type code:</b> 498		
Element name	Element identifier	Element scalar type
Alert report disabled	1	Type: Boolean Size: 1 octet Classification: Static Accessibility: Read/write Default value : True Valid value set: True, False
Alert report priority	2	Type: Unsigned8 Size: 1 octet Classification: Static Accessibility: Read/write Default value : 0 Valid value set: 0 to 15
Alarm limit	3	Type: IEEE 754_Float32 Size: 4 octets Classification: Static Accessibility: Read/write Default value : Not specified Valid value set : See IEEE 754 specification

**12.16.2.9 Binary alarm descriptor**

The binary alarm descriptor is the same structure as the data type for the alert report descriptor, so no additional data type description is required.

**12.16.2.10 ObjectIDandType**

The elements of ObjectIDandType are shown in Table 271.

**Table 271 – Data type: ObjectIDandType**

<b>Standard data type name: ObjectIDandType</b>		
<b>Standard data type code: 472</b>		
Element name	Element identifier	Element scalar type
ObjectID	1	Unsigned16
ObjectType	2	Unsigned8
ObjectSubType	3	Unsigned8
VendorSubType	4	Unsigned8

### 12.16.2.11 Unscheduled correspondent

The elements of Unscheduled Correspondent are shown in Table 272.

**Table 272 – Data type: Unscheduled correspondent**

<b>Standard data type name: Unscheduled Correspondent</b>		
<b>Standard data type code: 473</b>		
Element name	Element identifier	Element scalar type
Address	1	Unsigned128
TL Port	2	Unsigned16

## 12.17 Application services provided by application sub-layer

### 12.17.1 General

Application services are provided by the ASL (at the ASLDE-n SAP) for communication with native objects, which are either UAP objects or MP objects. These are the only services that may be used for compliance. Not all devices will need to use all of the services defined herein, and not all objects will support all the services herein. However, if these services are employed for communication between or among native objects, they are to be employed as defined in the standard.

Application processes using ASL services shall be designed to tolerate receipt of duplicate ASL service indications and confirmations. For example, if a lower layer acknowledgment is lost when a response to a read request is sent, the lower layer may retry, and as a result the application client may receive a duplicate response to the read request.

NOTE 1 It is left to the device to determine how best to handle congestion/back pressuring if locally indicated by the local lower protocol suite. This handling may, for example, limit transmission of messages from the device for a certain period of time, or for a certain set of communication priorities, or both. Congestion may occur, for example, in situations of network communication load, and handling by the device is intended to limit additional congestion.

NOTE 2 Capacity planning is a systems issue and is outside the scope of AL consideration.

Table 273 summarizes the services provided.

NOTE 3 Local services that do not result in network communication are not included in Table 273, as they are local matters and hence implementation-dependent.

NOTE 4 Local ASL service confirmation back to the AP is a local matter, and hence is not defined by this standard.

**Table 273 – AL services**

<b>ASL-provided service</b>	<b>Applicable primitives</b>	<b>Description</b>	<b>How used</b>
<b>Object access services</b>			
Read	Request Indication Response Confirmation	Read an attribute value from an object	Client/server
Write	Request Indication Response Confirmation	Write an value to an object	Client/server
Execute	Request Indication Response Confirmation	Execute a method on an object	Client/server
<b>Publication services</b>			
Publish	Request Indication	Publish a single or multiple values from one source object	Publish/subscribe  NOTE Native content, as well as non-native content, is supported.
<b>Alert report-related services</b>			
AlertReport	Request Indication	Report an alert	Source/sink (unicast only) The source of this service shall only be the ARMO. The sink of this service shall only be the alarm receiving object
AlertAcknowledge	Request Indication Response Confirmation	Acknowledge an individual alert reception	Client/server The source of this service may only be an alarm receiving object
<b>Explicit support for tunneling</b>			
Tunnel	Request Indication Response Confirmation	Tunnel payload without ASL parsing (for non-native protocol compatibility)	Tunnel payload without ASL parsing (for non-native protocol compatibility) This service shall be used by only if the source and destination objects are both tunnel objects

NOTE 5 If a local service request is malformed, reporting the error to the requesting UAP is a local matter, and thus is not addressed in the standard. If desired, it is possible for an implementation to keep statistics regarding locally received services requests that are malformed.

### 12.17.2 Publish/subscribe application communication model

Publication is a communication process that is initiated by an object in the publishing UAL and received by an object in the subscribing UAL. Publication uses ASL services specific to supporting publication. Publication occurs from a publisher object to a subscribing object. Any object may act as publisher or subscriber. To optimize communication bandwidth usage, special objects (a concentrator object for publishing and a dispersion object for subscribing) are defined to enable publication from/to a set of objects within a single UAP using a single publish service invocation.

The semantic reason for publishing is purely an application concern. This model supports communication for:

- Schedule-triggered periodic buffered communications;
- Application-triggered buffered communications; and
- Change-of-state-triggered buffered communications.

All of the above published communications use the publish/subscribe communication flow paradigm. For scheduled periodic communications, subscriber applications may support timeout response methods to deal with a loss of individual publications and/or a loss of the publisher endpoint. For example, a subscribing application may use prior publication value content, but may degrade the corresponding quality of the value. Loss of individual messages

may have a shorter timeout than the timeout used by the subscriber to determine the loss of the publisher.

Coordination of a publication with the network schedule is accomplished via appropriate endpoint configuration, which drives communication contract requests. A communication contract request is used to request that the system manager allocate scheduled bandwidth for publish communication.

NOTE 1 Determination of actual timeout policy when an expected publication is not received is an application process-specific matter that is not specified by this standard.

NOTE 2 Published messages with native content always contain a sequence counter and the current data value(s). If there is no change in value, the sequence counter shall indicate that the publishing application is still operating and has no new data to report (this is also known as a heartbeat). Some devices may retain this sequence counter to determine if the value has changed in order to limit reprocessing, while other devices may elect to ignore the sequence counter.

NOTE 3 For scheduled periodic communications, application processes may use common network time to synchronize their activities across the network. This synchronization may be locally applied by publishers and subscribers to synchronize their activities to the publication schedule.

NOTE 4 Continuous data and measurement in this standard employ a control system field proven publish/subscribe communication model and leverage use of scheduled bandwidth for more precise communication timing. If there is a more customized communication need, it is considered a custom situation outside the scope of this standard, or possibly a situation for future consideration.

### **12.17.3 Scheduled periodic buffered communication**

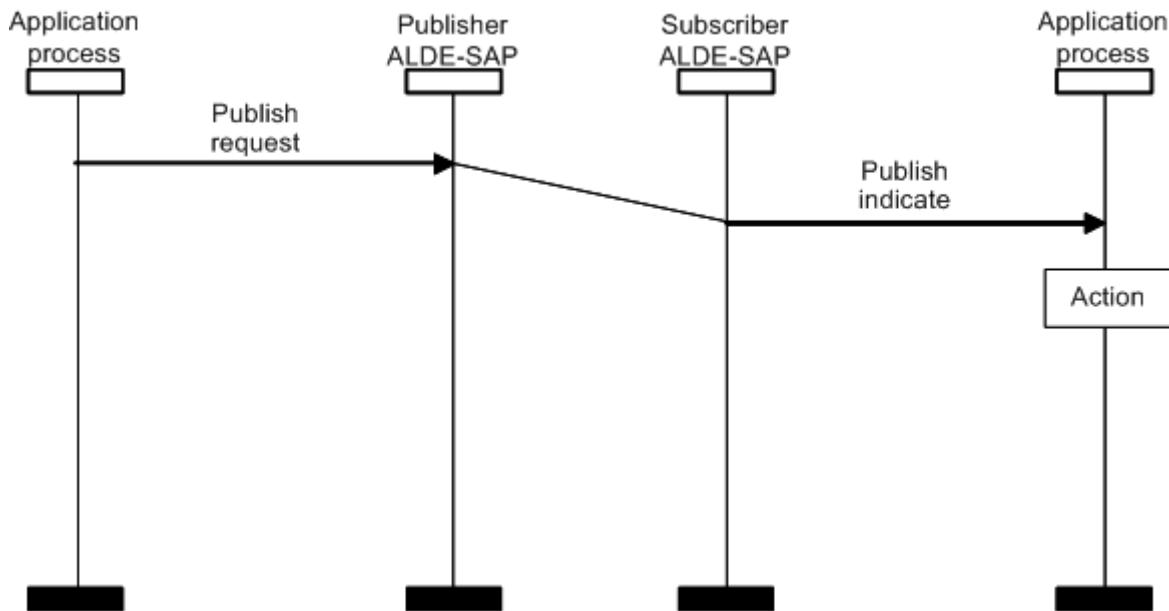
#### **12.17.3.1 General**

The publish service is used for unidirectional buffered communication to at most one subscriber.

NOTE 1 Publishing should be configured in accordance with the capabilities of the system. If a subscriber is not present, it is generally an interim or error situation and is not expected to be long-lived. Potential energy loss due to improperly configured or failed devices may be addressed by reconfiguration or device replacement. These are rare abnormal situations for which design optimization is not required; hence, the complexity required for publication only in the presence of an actual subscriber outweighs any communications savings.

No acknowledgements or retries are applied to publish/subscribe interactions. A publishing ASL is given an unconfirmed service request and constructs an appropriate APDU, which is then passed to the lower communication layers for communication transfer. If a publishing ASL receives a request for service before the prior request has been conveyed, the new request should overwrite the previous request and the previous request should not be transmitted. If a subscribing ASL receives a new request before the previous request was delivered to the destination object, the new request overwrites the previous request, and the previous request is lost.

The defined services support publication of an arbitrary attribute (for example, a process variable) from a simple device, or publication of a set of attributes from a more complex (for example, a multi-variable capable) wireless device. Publish/subscribe communications take place over a configured communication relationship, as shown in Figure 125.



**Figure 125 – Publish sequence of service primitives**

The subscriber for a publication may be, for example, a gateway.

NOTE 2 The content of non-native publications is outside the scope of the standard.

Native publications include the attribute value itself, and status information for the value. In order to support duplicate detection and out-of-order delivery, a simple one-octet monotonic counter is included with each published value.

NOTE 3 Other schemes for uniquely identifying a published message, such as using a timestamp instead of a counter, were considered but eliminated because they require more power to communicate. Time of TPDU construction, combined with a lower layer-provided freshness indication, may be locally available if needed. Timestamps on data publications are useful in remote terminal unit (RTU)-style buffering gateways, but for such gateways, a reception timestamp can be used for that purpose, particularly since all publications use a timeslot hopping schedule, leaving a small variance between data generation time and data reception time.

NOTE 4 Publish/subscribe message priority may be fixed; in that case, a local implementation may elect not to provide per-message priority. If an application requires publications with different priority levels, such as low priority publication for control monitoring and high priority publication for safety variables, separate publish/subscribe relationships are required.

The AL publisher and AL subscriber do not communicate explicitly to establish or break their relationship; however, establishment of secure communication relationships may force transport relationships to be established. Publishers and subscribers each establish their portion of the relationship independently (asynchronously). That is, either the publishing UAP or any of the subscribing UAPs may act first to establish its part of a publish/subscribe relationship. Both the publisher and the subscriber(s) shall establish their respective sides of the communication relationship in order for end-to-end messaging to commence. Once the publisher creates one side of the communication relationship and a subscriber creates the other side of the communication relationship, publication messages can be sent and delivered to the corresponding application objects.

NOTE 5 Locating publishers dynamically using either a tag discovery service or a centralized directory lookup service is outside the scope of this standard. Therefore, publish/subscribe is intended to be compatible with a static configuration mechanism. In future releases, a discovery mechanism may be employed. In either situation, the same information needs to be available to establish the publish/subscribe relationship.

Communication routes are formed transparently to the AL.

NOTE 6 The formation of transmission routes in support of publish/subscribe is important to ensure timely delivery, but is left as a responsibility for the system manager, which forms routes to use during communication contract establishment. See 6.3.11 for further details.

Publication is always an unconfirmed data transfer service request. Subscription always results in receiving an unconfirmed data transfer service indication.

NOTE 7 It is considered a management topic to ensure security configuration / changes support publish/subscribe without AL impact. It is understood that security considerations may constrain permitted relationships.

The timing of a scheduled publication is coordinated across the network, and as such depends on a coordinated view of time across the network. Bandwidth allocated for schedule-triggered publications needs to be reserved to ensure that subscribers can receive what their publishers send. The schedule should be configured to ensure best effort to meet delivery deadlines, but ultimately, the responsibility of the publisher to create new publications, and the subscriber to act on receipt of them, depends on the device's internal scheduling of the application process.

Published communications rely on the publication service support provided by the communication protocol suite that is underlying the ASL. An important aspect of this lower communication protocol suite is the ability to provide specific communication timing in order to meet scheduling demands.

### 12.17.3.2 Publish

#### 12.17.3.2.1 General

The publish service for this standard is a unicast service used to update data periodically from a single publication source in a single AP to (at most) a single subscriber destination.

The publish service may also be used in an aperiodic manner to support both application-triggered and change-of-state-triggered changes. Since buffer content is transmitted according to a schedule, native published communication includes a freshness indicator for to enable the subscriber to determine whether or not a value has changed.

NOTE Freshness does not mean unchanged data, but rather that the value has been newly (freshly) acquired since it was last published.

Table 274 defines the service primitives.

**Table 274 – Publish service**

Parameter name	Request	Indication
Argument	M	M
Service contract identifier	M	
Priority	M	
Discard eligible	M	
End-to-end transmission time		M
Published data size	M	M
Subscriber transport layer port	M	
Subscriber TDSAP		M
Subscribing object identifier	M	M(=)
Publisher network address		M
Publisher TDSAP	M	
Publisher transport layer port		M(=)
Publishing object identifier	M	M(=)
DataStructureInformation	M	M(=)
NativeIndividualValue	S	S(=)
Freshness sequence number	M	M(=)
Individual analog value and status	S	S(=)
Individual digital value and status	S	S(=)
NativeValueList	S	S(=)
Publishing content version	M	M(=)
List of publish data	M	M(=)
Fresh value sequence number	M	M(=)
Analog value and status	S	S(=)
Digital value and status	S	S(=)
Non-native	S	S(=)
Non-native data	M	M(=)

**12.17.3.2.2 Argument****12.17.3.2.2.1 Service contract identifier**

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

**12.17.3.2.2.2 Priority**

This parameter defines the message priority of service that is required of the communication. The permitted values for this service parameter may be an indication of either a high priority message or a low priority message. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

**12.17.3.2.2.3 Discard eligible**

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

**NOTE** This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

**12.17.3.2.2.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device. The interval is marked by two instants, the first instant being delivery to the transport layer in the requesting device, and the second instant being receipt by the transport layer in the destination device.

**12.17.3.2.2.5 Published data size**

This parameter provides the subscriber with the number of octets of the data to publish.

**12.17.3.2.2.6 Subscriber transport layer port**

This parameter identifies the subscriber UAP's associated transport layer port.

**12.17.3.2.2.7 Subscriber transport layer data service access point**

This parameter identifies the subscriber TDSAP associated with the subscriber transport layer port.

**12.17.3.2.2.8 Subscribing object identifier**

This parameter specifies the object identifier destination in the application that is subscribed to this publication.

**12.17.3.2.2.9 Publisher network address**

This identifies the 128-bit network address of the publisher.

**12.17.3.2.2.10 Publisher transport layer data service access point**

This parameter uniquely identifies the publisher UAPs associated TDSAP. The TDSAP maps 1-to-1 to a UAP. The value shall be a member of the set of valid TDSAPs, as specified by the transport layer.

**12.17.3.2.2.11 Publisher transport layer port**

This parameter identifies the publisher UAPs associated transport layer port.

**NOTE** An implementation may infer this parameter from the publisher TDSAP. It is included for completeness, as required by this standard for the logical mapping to the transport data service request definition.

**12.17.3.2.2.12 Publisher object identifier**

This parameter identifies the publisher object that is the source of the published data.

NOTE If there is more than one entry in the list of published data, the publishing object source is an instance of the concentrator object type.

#### **12.17.3.2.2.13 Data structure information**

This parameter indicates the construct of the information to be conveyed via publication. It may indicate one of the following constructs:

- Native individual value;
- Native sequence of values; or
- Non-native data (that is, information being tunneled via a publication service).

#### **12.17.3.2.2.14 Native individual value**

##### **12.17.3.2.2.14.1 General**

This clause defines the data structure of information for a single native value included in the production.

##### **12.17.3.2.2.14.2 Freshness value sequence number**

This parameter is present if the data structure information indicates the structure of the data is a native individual value. This parameter indicates the freshness of the data.

##### **12.17.3.2.2.14.3 Individual analog value and status**

This parameter is present if the individual native value is an analog. This contains standard value status data structure that indicates information such as quality of the corresponding analog value and the analog value itself.

##### **12.17.3.2.2.14.4 Individual digital value and status**

This parameter is present if the individual native value is digital. This contains standard value status data structure that indicates information such as quality of the corresponding digital value and the digital value itself.

#### **12.17.3.2.2.15 Native value list**

##### **12.17.3.2.2.15.1 General**

This clause defines the data structure of information for a list of native values included in the production.

##### **12.17.3.2.2.15.2 Publishing content version**

This parameter is present if the publication is for native list data, such as sent from a concentrator object. This information ensures harmonious interpretation of the published information by the subscriber.

##### **12.17.3.2.2.15.3 List of publish data**

###### **12.17.3.2.2.15.3.1 General**

This parameter represents the list of data conveyed via the publish service.

###### **12.17.3.2.2.15.3.2 Status and analog value**

This contains standard value status data structure that indicates information such as quality of the corresponding analog value and the analog value itself.

###### **12.17.3.2.2.15.3.3 Status and digital value**

This contains standard value status data structure that indicates information such as quality of the corresponding digital value and the digital value itself.

#### **12.17.3.2.2.16 Non-native**

##### **12.17.3.2.2.16.1.1 General**

This represents the data contained in the publish service that is non-native.

### 12.17.3.2.2.16.1.2 Non-native data

This parameter contains the non-native data to publish. Non-native data is conveyed as a string of octets.

### 12.17.4 Client-server interactions

#### 12.17.4.1 General

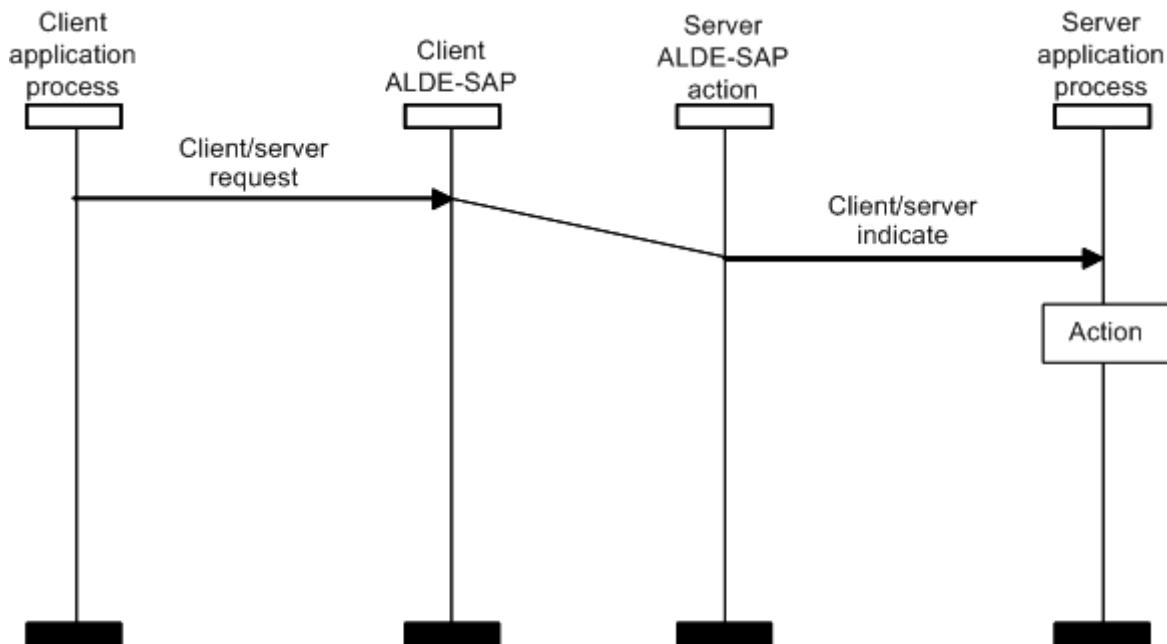
Client/server interactions are used for one-to-one aperiodic communications. These relationships employ on-demand queued bidirectional communication. Client/server services defined by this standard are either two-part service (having two service primitives, .req and .ind), as in Figure 126, or four-part service (having four service primitives, .req, .ind, .rsp, and .cnf), as in Figure 127, Figure 128 and Figure 129.

When the ASL receives a client/server service request, it constructs a corresponding application protocol data unit (APDU) and requests queued transfer from the lower communication protocol suite. The server ASL is given a confirmed service response indication, which it delivers to the destination UAP and object.

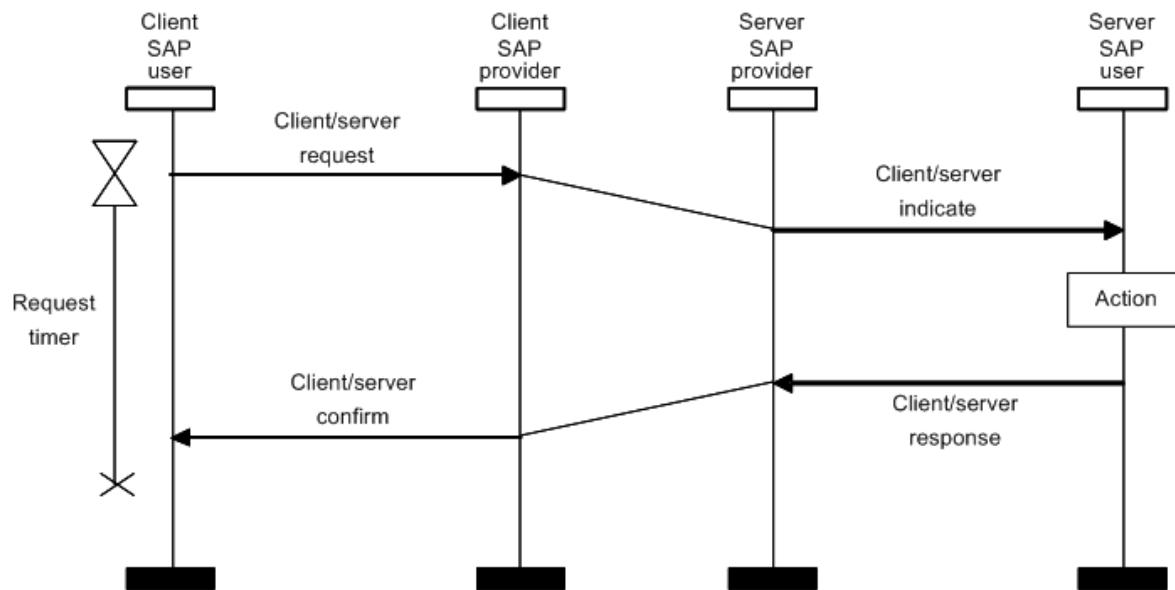
For services with four-part primitives defined, the server UAP constructs a corresponding response. When the ASL receives the client/server service response, it constructs a response APDU and submits it to its lower communication protocol suite, which provides queue-oriented communication services to deliver the response.

In a client/server interaction, either endpoint of the communication can act as client or server or both. A client request is sent to a single destination (server). This request indicates the destination to which the response should be sent. A single response is then issued from the server.

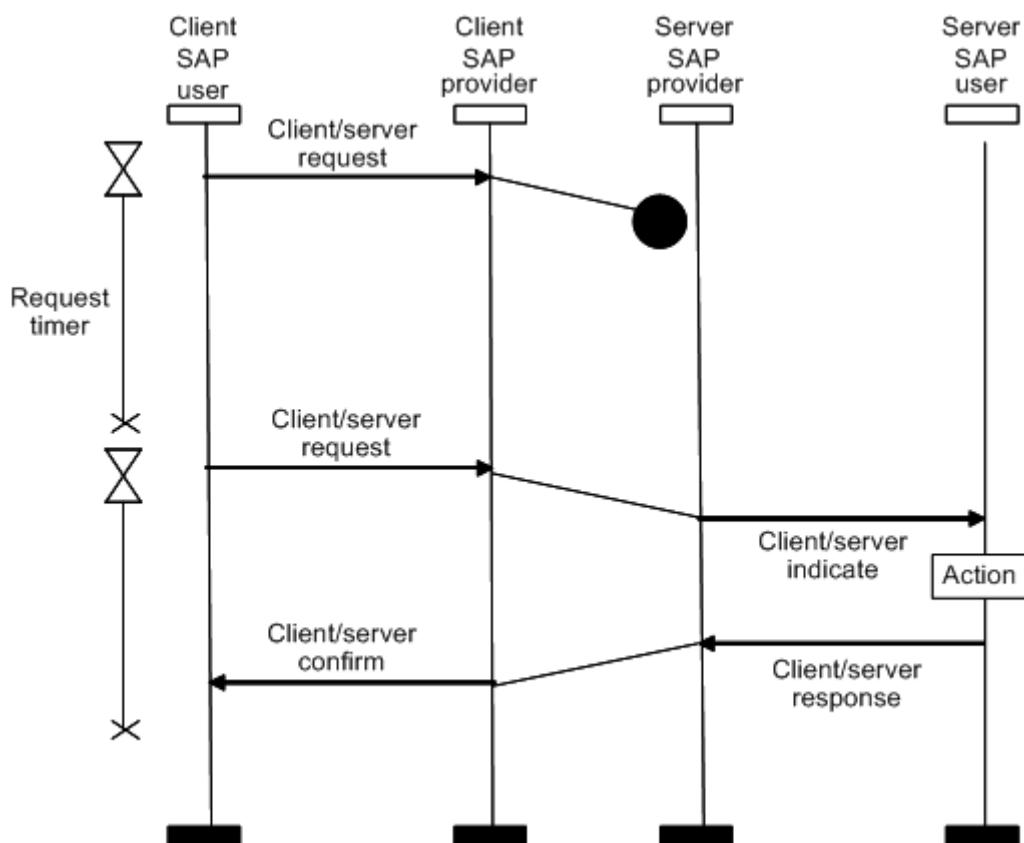
Interactions as shown in Figure 127, Figure 128, and Figure 129 require a communication contract identifier and the communication protocol suite to be appropriately configured to support the client/server messaging requirements of the application. The bandwidth represented by the contract identifier is considered unscheduled shared bandwidth which need not be reserved solely for use by this contract.



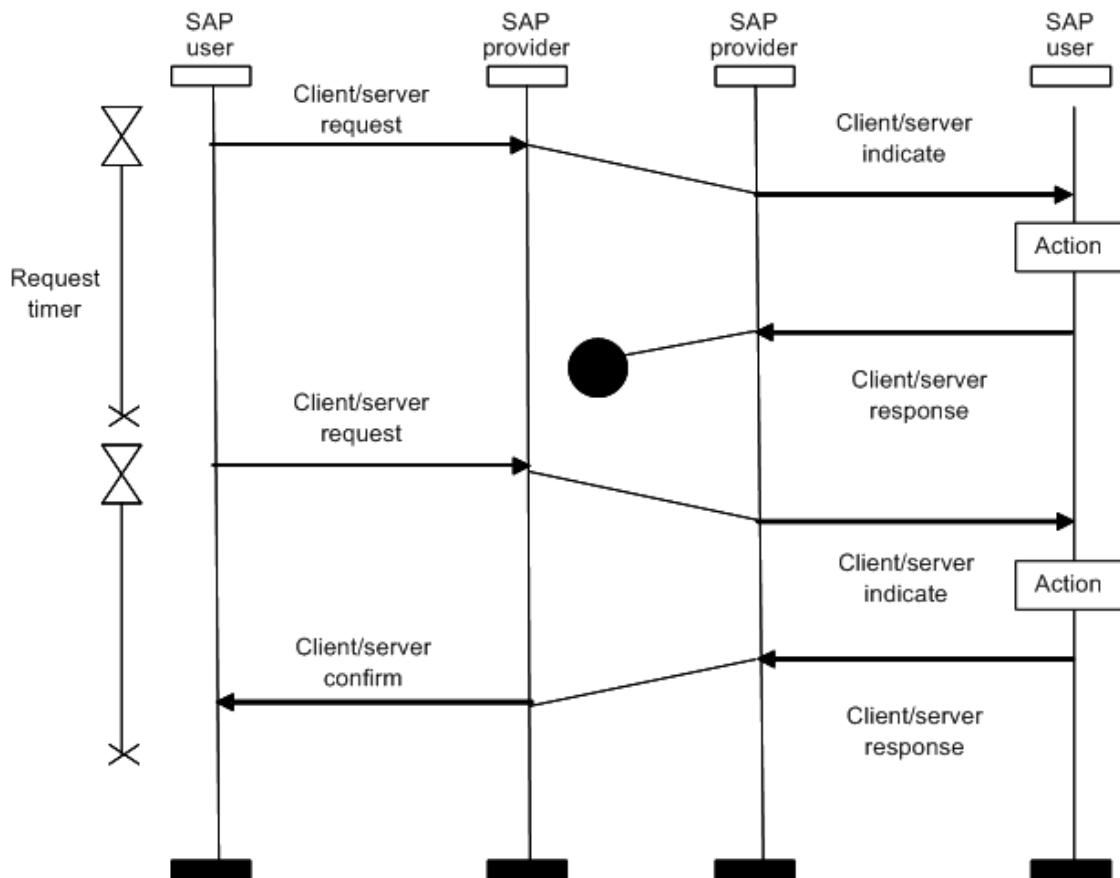
**Figure 126 – Client/server model two-part interactions**



**Figure 127 – Client/server model four-part interactions: Successful delivery**



**Figure 128 – Client/server model four-part interactions: Request delivery failure**



**Figure 129 – Client/server model four-part interactions: Response delivery failure**

NOTE 1 When a client/server association is secured, a security session is involved. To optimize elimination of connections that the UAL process knows are no longer required, a local interface to terminate the contract may be employed. Contract termination may be used to release a security session (if one exists).

To initiate communication, the client requests to send a message to a server. The client specifies the local contract identifier which indicates the server's network address. The communication also identifies the particular source application and object making the request, the destination application and object intended to receive the request, as well as the actual service instance specific request information.

The server sends a response to the client specifying its local contract identifier, which indicates the client's network address. The communication also specifies sufficient information to deliver the response to the appropriate application object and to collate the response with the original request.

A communication contract shall be established between client and server to carry the client's request, and correspondingly from the server to the client to carry the server response.

Acting in the role of a client, a client may send requests to a server. Acting in the role of a server, a server may send responses to a client. The client is responsible for server timeout response and transactional integrity.

A simple server might support as few as one outstanding transaction with a particular client. If an extended delay occurs in receiving a response, the client may, for example, timeout and resend the request. If this occurs, duplicate responses may be received. If a server has resources to support multiple outstanding transactions with a client, requests and responses may arrive out of order. To support this situation, a request identifier is used to enable request/response collation.

NOTE 2 If there is a need for multiple client or multiple server messages as part of a communication sequence, the implementation may consider employing ASL concatenation. Beyond concatenation, streaming of messages is an application process-specific responsibility outside the scope of the standard.

Communications characteristics for client/server interactions such as response timeout are local matters, beyond the scope of this standard. For example, they may be fixed by a device or application within a device, or configured on the device, on a per-application process basis or even a per contract basis. Client/server interactions are typically used for configuration (such as process control related configuration or management object configuration.) and ad-hoc exchange of information.

NOTE 3 Client/server communications should not interfere with scheduled communications as it is required that transmission bandwidth be allocated to support client/server messaging communication contracts. The intent is to ensure the ability to reconfigure a device.

Occasionally, only a single client/server exchange is required. This may entail substantial overhead in route and security establishment. At other times, multiple client/server exchanges occur between the same endpoints.

NOTE 4 Any required alteration of communication routes (for example, to compensate for interference) occurs transparently to the AL.

The client specifies the desired message priority for service requests; the server specifies the desired message priority for service responses.

NOTE 5 Higher priority messages should ideally move to the front of prioritized communication protocol suite message queues supporting client/server communications. If possible, client/server message bandwidth allocation on the network should grant access first to higher priority message requests. Security is presumed to be on a contract basis, so per-message security is not provided.

NOTE 6 Further considerations for transmission back-off, such as based on network congestion are an overall device responsibility, and are not a specific responsibility of the AL.

## **12.17.4.2 Client/server services**

### **12.17.4.2.1 General**

The following services are provided as client/server communications:

- Read;
- Write;
- Execute; and
- Tunnel.

NOTE Tunnel as a two-part primitive is also useful for source/sink communication.

### **12.17.4.2.2 Service feedback codes**

Four-part client/server services provide a service feedback code to indicate the result of the service from the viewpoint of the server. A range of codes is reserved for vendor-specific additions.

### **12.17.4.2.3 Read**

#### **12.17.4.2.3.1 General**

The read service is used to read an attribute of an object from a UAL process.

Table 275 defines the read service primitives.

**Table 275 – Read service**

Parameter name	Request	Indication	Response	Confirm
Argument	M	M		
Service contract identifier	M			
Priority	M			
Discard eligible	M			
End-to-end transmission time		M		
Forward congestion notification		M		
Server transport layer port	M			
Server TDSAP		M		
Server object identifier	M	M(=)		
Client network address		M		
Client TDSAP	M			
Client transport layer port		M		
Client object identifier	M	M(=)		
Application request ID	M	M(=)		
Data to be read	M	M(=)		
Attribute identifier	M	M(=)		
Attribute index(es)	C	C(=)		
Result			M	M
Service contract identifier			M	
Priority			M	
Discard eligible			M	
End-to-end transmission time				M
Forward congestion notification				M
Forward congestion notification echo			M	M(=)
Server network address				M
Server TDSAP			M	
Server transport layer port				M
Server object identifier			M	M(=)
Client transport layer port			M	
Client TDSAP				M
Client object identifier			M	M(=)
Application request ID			M	M(=)
Value read			M	M(=)
Service feedback code			M	M(=)
Value size			C	C(=)
Data value			C	C(=)

#### 12.17.4.2.3.2 Argument

##### 12.17.4.2.3.2.1 Service contract identifier

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

##### 12.17.4.2.3.2.2 Priority

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

##### 12.17.4.2.3.2.3 Discard eligible

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

NOTE This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

#### **12.17.4.2.3.2.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

#### **12.17.4.2.3.2.5 Forward congestion notification**

This parameter indicates if the request has encountered network congestion on its path from the client to the server.

#### **12.17.4.2.3.2.6 Server transport layer port**

This parameter identifies the server UAP's associated TL port.

#### **12.17.4.2.3.2.7 Server transport layer data service access point**

This parameter identifies the server TDSAP associated with the server transport layer port.

#### **12.17.4.2.3.2.8 Server object identifier**

This parameter identifies a server object from which data is desired to be read.

#### **12.17.4.2.3.2.9 Client/source address**

This parameter identifies the network address for the client of this request.

#### **12.17.4.2.3.2.10 Client/source transport layer data service access point**

This parameter identifies the client or source UAP's associated TDSAP. The UAP contains the object originating the request.

#### **12.17.4.2.3.2.11 Client transport layer port**

This parameter identifies the client UAP's associated transport layer port.

#### **12.17.4.2.3.2.12 Client object identifier**

This parameter identifies the client object that is initiating the service request.

#### **12.17.4.2.3.2.13 Application request identifier**

An identifier provided by the UAP to uniquely represent this request.

#### **12.17.4.2.3.2.14 Data to be read**

This parameter identifies the data values that the client desires to read.

#### **12.17.4.2.3.2.15 Attribute identifier**

This parameter identifies the attribute of the server object, the value of which is desired to be read.

#### **12.17.4.2.3.2.16 Attribute index/indices**

This parameter identifies the index/indices for the information of interest from the attribute. There may be:

- Zero indices, such as for a scalar value;
- One index, for example, to access an element of a singly-dimensioned array or an element of a standard data structure; or
- Two indices, for example, to access an element of a doubly-dimensioned array or to access an element of a standard structure contained in a singly-dimensioned array of standard data structures.

**12.17.4.2.3.3 Result****12.17.4.2.3.3.1 Service contract identifier**

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

**12.17.4.2.3.3.2 Priority**

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

**12.17.4.2.3.3.3 Discard eligible**

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

NOTE This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

**12.17.4.2.3.3.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

**12.17.4.2.3.3.5 Forward congestion notification**

This parameter indicates if the service response encountered network congestion on its path from the server to the client.

**12.17.4.2.3.3.6 Forward congestion notification echo**

This parameter indicates if the service request encountered network congestion on its path from the client to the server.

**12.17.4.2.3.3.7 Server network address**

This parameter identifies the network address for the server for this request.

**12.17.4.2.3.3.8 Server transport layer data service access point**

This parameter identifies the server UAP's associated TDSAP.

**12.17.4.2.3.3.9 Server transport layer port**

This parameter identifies the server UAP's associated transport layer port.

**12.17.4.2.3.3.10 Server object identifier**

This parameter identifies a server object from which data is desired to be read.

**12.17.4.2.3.3.11 Client transport layer port**

This parameter identifies the client or source UAP's associated TL port. The UAP contains the object originating the request.

**12.17.4.2.3.3.12 Client transport layer data service access point**

This parameter identifies the client TDSAP associated with the transport layer port. The UAP contains the object originating the request.

**12.17.4.2.3.3.13 Client object identifier**

This parameter identifies the client object that is initiating the service request.

**12.17.4.2.3.3.14 Application request identifier**

This parameter is an identifier provided by the client to uniquely represent this request.

**12.17.4.2.3.3.15 Value read**

The value read indicates the result of the read operation, and if the read was successful, the size and value of the object attribute to be read.

**12.17.4.2.3.3.16 Service feedback code**

The service feedback code indicates if the read operation was successful or not. If not successful, it provides information indicating why it was not successful.

**12.17.4.2.3.3.17 Value size**

Value size indicates the number of octets contained in data value. It is present if and only if the corresponding service feedback code indicates success.

**12.17.4.2.3.3.18 Data value**

Data value is the data value that was read from the identified server object, attribute, and attribute index. It is present if and only if the corresponding service feedback code indicates success, and the Value size is non-zero.

**12.17.4.2.4 Write****12.17.4.2.4.1 General**

The write service is used to write a value or set of values to one or more attribute of one or more objects in an application process.

A write to a structure containing both writeable and read-only elements is permitted. In this situation, the read-only elements shall be unaffected.

Table 276 defines the write service primitives.

**Table 276 – Write service**

Parameter name	Request	Indication	Response	Confirm
Argument	M	M		
Service contract identifier	M			
Priority	M			
Discard eligible	M			
End-to-end transmission time		M		
Forward congestion notification		M		
Server transport layer port	M			
Server TDSAP		M		
Server object identifier	M	M(=)		
Client network address		M		
Client TDSAP	M			
Client transport layer port		M		
Client object identifier	M	M(=)		
Application request ID	M	M(=)		
Data to write	M	M(=)		
Attribute identifier	M	M(=)		
Attribute index(es)	M	M(=)		
Value size	M	M(=)		
Data value	M	M(=)		
Result			M	M
Service contract identifier			M	
Priority			M	
Discard eligible			M	
End-to-end transmission time				M
Forward congestion notification				M
Forward congestion notification echo			M	M(=)
Server network address				M
Server TDSAP			M	
Server transport layer port				M(=)
Server object identifier			M	M(=)
Client transport layer port			M	
Client TDSAP				M
Client object identifier			M	M(=)
Application request ID			M	M(=)
Service feedback code			M	M(=)

#### 12.17.4.2.4.2 Argument

##### 12.17.4.2.4.2.1 Service contract identifier

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

##### 12.17.4.2.4.2.2 Priority

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

##### 12.17.4.2.4.2.3 Discard eligible

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

NOTE This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

**12.17.4.2.4.2.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

**12.17.4.2.4.2.5 Forward congestion notification**

This parameter indicates if the request has encountered network congestion on its path from the client to the server.

**12.17.4.2.4.2.6 Server transport layer port**

This parameter identifies the server UAP's associated TL port.

**12.17.4.2.4.2.7 Server transport layer data service access point**

This parameter identifies the server TDSAP associated with the server TL port.

**12.17.4.2.4.2.8 Server object identifier**

This parameter identifies a server object from which data is desired to be written.

**12.17.4.2.4.2.9 Client/source address**

This parameter identifies the network address for the client of this request.

**12.17.4.2.4.2.10 Client/source transport layer data service access point**

This parameter identifies the client or source UAP's associated TDSAP. The UAP contains the object originating the request.

**12.17.4.2.4.2.11 Client transport layer port**

This parameter identifies the client UAP's associated TL port.

**12.17.4.2.4.2.12 Client object identifier**

This parameter identifies the client object that is initiating the service request.

**12.17.4.2.4.2.13 Application request identifier**

An identifier provided by the UAP to uniquely represent this request.

**12.17.4.2.4.2.14 Data to write**

This parameter identifies the target attribute and data value that the client desires to write.

**12.17.4.2.4.2.15 Attribute identifier**

This parameter identifies the attribute of the server object, the value of which is desired to be read.

**12.17.4.2.4.2.16 Attribute index/indices**

This parameter identifies the index/indices for the information of interest from the attribute. There may be:

- Zero indices, such as for a scalar value;
- One index, for example, to access an element of a singly-dimensioned array or an element of a standard data structure; or
- Two indices, for example, to access an element of a doubly-dimensioned array, or to access an element of a standard structure contained in a singly-dimensioned array of standard data structures.

**12.17.4.2.4.2.17 Value size**

Value size indicates the number of octets contained in data value.

**12.17.4.2.4.2.18 Data value**

Data value is the data value that is desired to be written to the identified server object, attribute, and attribute index.

**12.17.4.2.4.3 Result****12.17.4.2.4.3.1 Service contract identifier**

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

**12.17.4.2.4.3.2 Priority**

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

**12.17.4.2.4.3.3 Discard eligible**

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

**NOTE** This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

**12.17.4.2.4.3.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

**12.17.4.2.4.3.5 Forward congestion notification**

This parameter indicates if the service response encountered network congestion on its path from the server to the client.

**12.17.4.2.4.3.6 Forward congestion notification echo**

This parameter indicates if the service request encountered network congestion on its path from the client to the server.

**12.17.4.2.4.3.7 Server network address**

This parameter identifies the network address for the server for this request.

**12.17.4.2.4.3.8 Server transport layer data service access point**

This parameter identifies the server UAP's associated TDSAP.

**12.17.4.2.4.3.9 Server transport layer port**

This parameter identifies the server UAP's associated TL port.

**12.17.4.2.4.3.10 Server object identifier**

This parameter identifies a server object to which data is desired to be written.

**12.17.4.2.4.3.11 Client/source transport layer port**

This parameter identifies the client UAP's associated TL port.

**12.17.4.2.4.3.12 Client transport layer data service access point**

This parameter identifies the client TDSAP associated with the TL port. The UAP contains the object originating the request.

**12.17.4.2.4.3.13 Client object identifier**

This parameter identifies the client object that is initiating the service request.

#### **12.17.4.2.4.3.14 Application request identifier**

An identifier provided by the client to uniquely represent this request.

#### **12.17.4.2.4.3.15 Service feedback code**

The service feedback code indicates if the write operation was successful or not. If not successful, it provides information indicating why it was not successful.

#### **12.17.4.2.5 Execute**

##### **12.17.4.2.5.1 General**

The execute service is used to execute a network visible method on an object.

NOTE One use of the execute service may be to establish a callback method. This ensures that a server has adequate time to provide information back to the client via a callback, rather than via providing execution results in the response.

Table 277 defines the execute service primitives.

**Table 277 – Execute service**

Parameter name	Request	Indication	Response	Confirm
Argument	M	M		
Service contract identifier	M			
Priority	M			
Discard eligible	M			
End-to-end transmission time		M(=)		
Forward congestion notification		M		
Server transport layer port	M			
Server TDSAP		M		
Server object identifier	M	M(=)		
Client network address		M		
Client TDSAP	M			
Client transport layer port		M		
Client object identifier	M	M(=)		
Application request ID	M	M(=)		
Method to execute	M	M(=)		
Method identifier	M	M(=)		
Size of input parameters	M	M(=)		
Input parameters	C	C(=)		
Result			M	M
Service contract identifier			M	
Priority			M	
Discard eligible			M	
End-to-end transmission time				M
Forward congestion notification				M
Forward congestion notification echo			M	M(=)
Server network address				M
Server TDSAP			M	
Server transport layer port				M
Server object identifier			M	M(=)
Client transport layer port			M	
Client TDSAP				M
Client object identifier			M	M(=)
Application request ID			M	M(=)
Execution result			M	M(=)
Service feedback code			M	M(=)
Size of output parameters			M	M(=)
Output parameters			C	C(=)

**12.17.4.2.5.2 Argument****12.17.4.2.5.2.1 Service contract identifier**

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

**12.17.4.2.5.2.2 Priority**

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

**12.17.4.2.5.2.3 Discard eligible**

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

**NOTE** This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

**12.17.4.2.5.2.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

**12.17.4.2.5.2.5 Forward congestion notification**

This parameter indicates if the request has encountered network congestion on its path from the client to the server.

**12.17.4.2.5.2.6 Server transport layer port**

This parameter identifies the server UAP's associated TL port.

**12.17.4.2.5.2.7 Server transport layer data service access point**

This parameter identifies the server TDSAP associated with the server TL port.

**12.17.4.2.5.2.8 Server object identifier**

This parameter identifies a server object on which the method is to be executed.

**12.17.4.2.5.2.9 Client/source address**

This parameter identifies the network address for the client of this request.

**12.17.4.2.5.2.10 Client/source transport layer data service access point**

This parameter identifies the client or source UAP's associated TDSAP. The UAP contains the object originating the request.

**12.17.4.2.5.2.11 Client transport layer port**

This parameter identifies the client UAP's associated TL port.

**12.17.4.2.5.2.12 Client object identifier**

This parameter identifies the client object that is initiating the service request.

**12.17.4.2.5.2.13 Application request identifier**

An identifier provided by the UAP to uniquely represent this request.

**12.17.4.2.5.2.14 Method identifier**

This parameter identifies the method of the server object that is desired to be executed.

#### **12.17.4.2.5.2.15 Size of input parameters**

Size of input parameters indicates the number of octets contained in input parameters.

NOTE Execute requests and responses include the size in octets of the contained parameter stream to enable parsing (this is especially useful in APDU concatenation scenarios).

#### **12.17.4.2.5.2.16 Input parameters**

The input parameters string is an octet string that contains the input parameters for the method that is being requested to be executed. This is present if and only if size of input parameters is present and has a value greater than zero.

#### **12.17.4.2.5.3 Result**

##### **12.17.4.2.5.3.1 Service contract identifier**

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

##### **12.17.4.2.5.3.2 Priority**

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

##### **12.17.4.2.5.3.3 Discard eligible**

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

NOTE This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

##### **12.17.4.2.5.3.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

##### **12.17.4.2.5.3.5 Forward congestion notification**

This parameter indicates if the service response encountered network congestion on its path from the server to the client.

##### **12.17.4.2.5.3.6 Forward congestion notification echo**

This parameter indicates if the service request encountered network congestion on its path from the client to the server.

##### **12.17.4.2.5.3.7 Server network address**

This parameter identifies the network address for the server for this request.

##### **12.17.4.2.5.3.8 Server transport layer data service access point**

This parameter identifies the server UAP's associated TDSAP.

##### **12.17.4.2.5.3.9 Server transport layer port**

This parameter identifies the server UAP's associated TL port.

##### **12.17.4.2.5.3.10 Server object identifier**

This parameter identifies a server object on which the method was to be executed.

##### **12.17.4.2.5.3.11 Client/source transport layer port**

This parameter identifies the client UAP's associated TDSAP.

**12.17.4.2.5.3.12 Client transport layer data service access point**

This parameter identifies the client TDSAP associated with the TL port. The UAP contains the object originating the request.

**12.17.4.2.5.3.13 Client object identifier**

This parameter identifies the client object that is initiating the service request.

**12.17.4.2.5.3.14 Application request identifier**

An identifier provided by the client to uniquely represent this request.

**12.17.4.2.5.3.15 Execution result**

This contains the result of the method execution service request.

**12.17.4.2.5.3.16 Service feedback code**

The service feedback code indicates if the corresponding method execution was successful or not. If not successful, it provides information indicating why it was not successful.

**12.17.4.2.5.3.17 Size of output parameters**

Size of output parameters indicates the number of octets contained in output parameters.

**12.17.4.2.5.3.18 Output parameters**

The output parameters string is an octet string that contains the output parameters for the method that was executed. This is present if and only if size of output parameters is present and has a value greater than zero.

**12.17.5 Unscheduled acyclic queued unidirectional messages (source/sink)****12.17.5.1 General**

Unscheduled acyclic queued unidirectional messaging is also sometimes referred to as source/sink messaging. This interaction type is used for alerts. Messages sent using this protocol are queued by the lower communication layers for transmission. Message receipt is unconfirmed. There is no application process flow or rate control or lost message detection for this mode of interaction. Like client/server communications, these communications require use of a communication contract, and specify message priority on a per-message basis.

Bandwidth for source/sink communications is not considered dedicated, but rather is considered to come from non-dedicated (i.e., shared) bandwidth.

Unscheduled acyclic unidirectional interactions in this standard support on-demand one-to-one queued message distribution. Alert reports for network communication are always issued from one initiator, the ARMO, and are always sent to one type of message recipient, an alert receiving object (ARO).

Acknowledgement of reception of an individual alert may only be issued from one alert report Recipient (ARO), and is sent to the (ARMO) object that reported the alert.

The following services are provided to support unscheduled acyclic queued unidirectional message communications:

- AlertReport;
- AlertAcknowledge; and
- Tunnel.

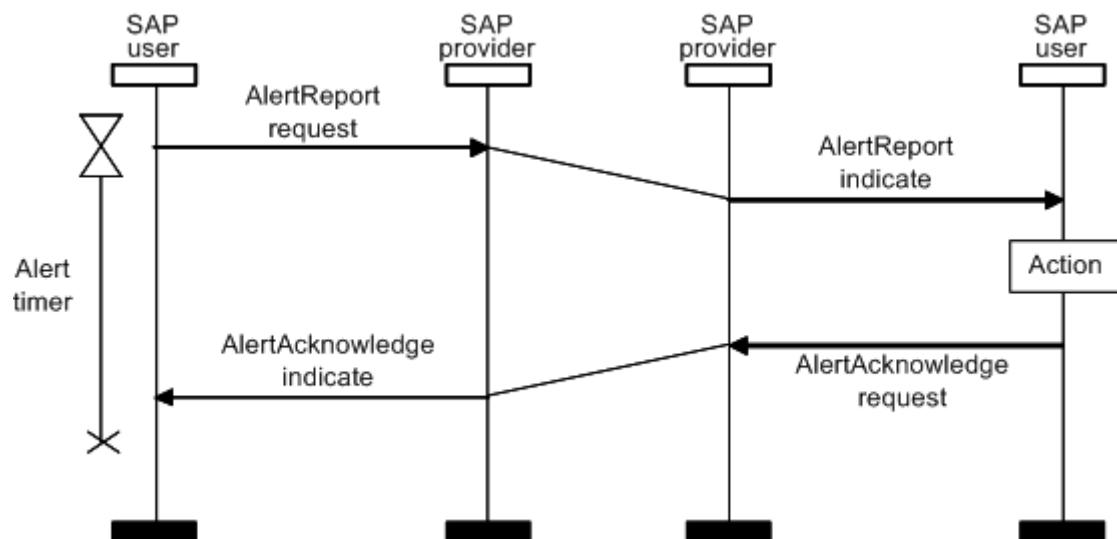
NOTE Tunnel is included as a source/sink service, since it may need to take advantage of multi-cast capabilities in future releases of this standard.

### 12.17.5.2 AlertReport service

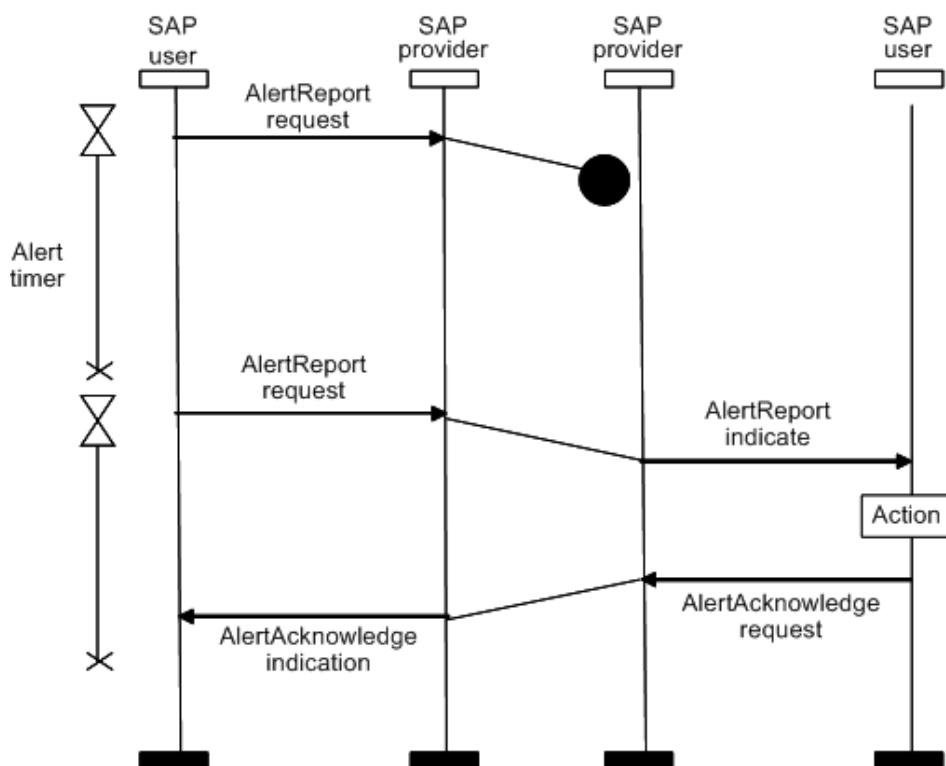
#### 12.17.5.2.1 General

AlertReport is used to report an alert using queued unidirectional communication services. The content of the alert report depends on the type of alert being reported and the category of the alert. AlertReports may be retried until an AlertAcknowledge for the AlertReport has been received.

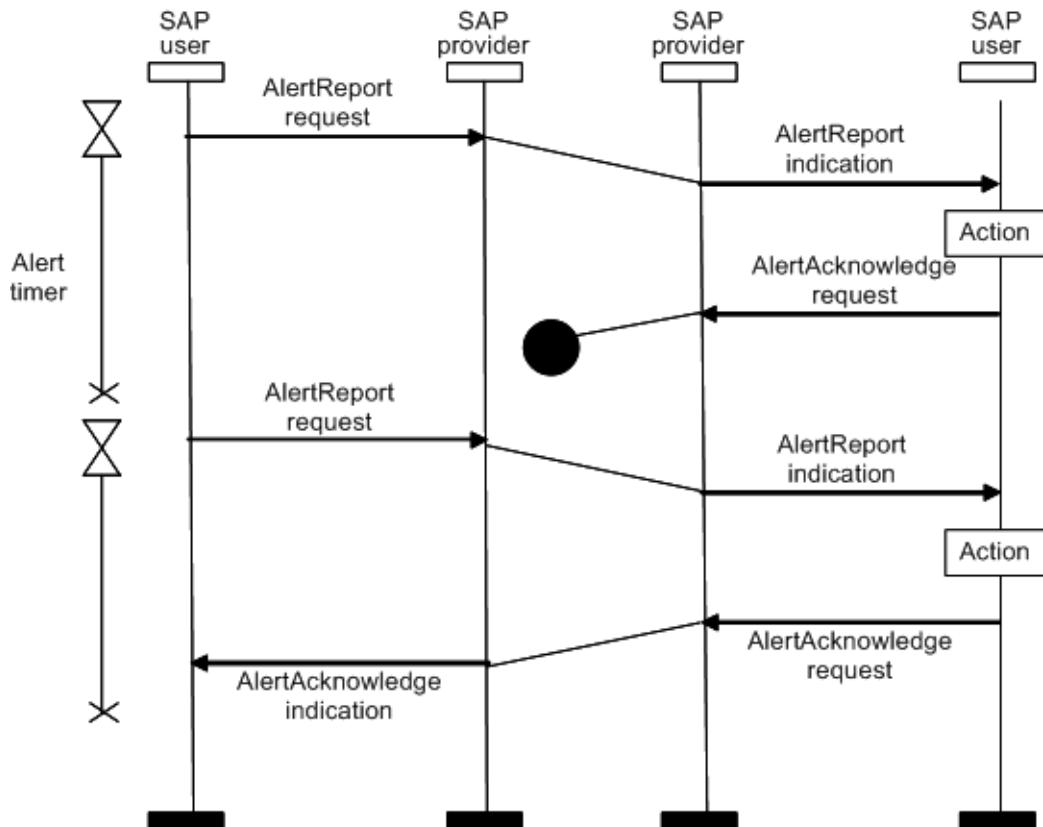
Figure 130, Figure 131, and Figure 132 indicate alert reporting message sequencing.



**Figure 130 – AlertReport and AlertAcknowledge, delivery success**



**Figure 131 – AlertReport, delivery failure**



**Figure 132 – Alert Report, acknowledgment failure**

NOTE AlertReport timeout/retry policy is defined on the ARMO. See 6.2.7.2 for further details.

### 12.17.5.2.2 AlertReport service primitives

#### 12.17.5.2.2.1 General

Alert reporting employs two separate two-part application communication services. To report an alert, the AP uses the AlertReport service. In this version of the standard, the recipient of an AlertReport shall acknowledge the AlertReport by using the AlertAcknowledge service.

NOTE The use of two separate services, AlertReport and AlertAcknowledgement is required in order to enable a single alert to be sent to multiple destinations in a future revision to this standard.

Monitoring and checking for acknowledgement, as well as re-reporting an alert condition for which acknowledgement has not been received is the responsibility of the alert reporting device. Re-reporting an alert that is no longer prevalent, and for which an AlertAcknowledge indication has not been received, is a local matter. For example, if a diagnostic situation occurs and an alert is reported, and then the reporting device reboots such that the diagnostic situation is no longer prevailing, the device might not re-report the diagnostic alert that was in effect prior to reboot even though no AlertAcknowledge was received.

AlertReport employs the same communication model as a two-part client/server primitive. Table 278 defines the service primitives for the AlertReport service.

**Table 278 – AlertReport service**

Parameter name	Request	Indication
Argument	M	M
Service contract identifier	M	
Priority	M	
Discard eligible	M	
End-to-end transmission time		M
ARMO TDSAP	M	
ARMO transport layer port		M
ARMO	M	M(=)
Sink transport layer port	M	
Sink TDSAP		M
Sink object identifier	M	M(=)
Individual alert report	M	M(=)
Individual alert identifier	M	M(=)
Alert detector transport layer port	M	M(=)
Alert detector object	M	M(=)
Detection time	M	M(=)
Alert class	M	M(=)
Alarm direction	C	C(=)
Alert category	M	M(=)
Alert priority	M	M(=)
Alert type	M	M(=)
Alert value size	M	M(=)
Alert value	O	O(=)

### 12.17.5.2.2.2 Argument

#### 12.17.5.2.2.2.1 Service contract identifier

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

#### 12.17.5.2.2.2.2 Priority

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

#### 12.17.5.2.2.2.3 Discard eligible

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

NOTE This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

#### 12.17.5.2.2.2.4 End-to-end transmission time

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

#### 12.17.5.2.2.2.5 Alert reporting management object transport layer data service access point

This parameter indicates the TDSAP of the application which is issuing this alert report.

#### 12.17.5.2.2.2.6 Alert reporting management object transport layer port

This parameter indicates the transport layer port of the application which is issuing this alert report.

**12.17.5.2.2.2.7 Alert reporting management object**

This parameter represents the object identifier of the ARMO that is reporting the alert.

**12.17.5.2.2.2.8 Sink transport layer port**

This parameter identifies the sink UAP's associated TL port.

**12.17.5.2.2.2.9 Sink transport layer data service access point**

This parameter indicates the TDSAP corresponding to sink transport layer port.

**12.17.5.2.2.2.10 Sink object identifier**

This parameter specifies the destination sink object in the application to which this service request is to be sent.

**12.17.5.2.2.2.11 Alert source network address**

This parameter identifies the network address of the source of this request.

**12.17.5.2.2.2.12 Alert source transport layer data service access point**

This parameter identifies the source UAP's associated TDSAP.

**12.17.5.2.2.2.13 Source transport layer port**

This parameter identifies the transmitting application source UAP associated with the transport layer port.

**12.17.5.2.2.2.14 Individual alert report**

This parameter contains an individual alert being reported by this service invocation.

**12.17.5.2.2.2.15 Individual alert identifier**

This parameter uniquely identifies the individual alert report. Separate identifier value sequences for the alert reporting categories shall be maintained. The value of this parameter shall be monotonically increasing, and shall wrap around when the maximum value is reached. It is included when an individual alert report is acknowledged. It also is used by an alert receiver to determine if an alert report or a set of alert reports of a particular category has / have been missed. If a missed report condition is detected, an alarm recovery operation should be performed. Refer to the Alarm\_Regen attributes of the ARMO in 6.2.7.2 for further details of triggering the regeneration.

**12.17.5.2.2.2.16 Alert source transport port**

This parameter identifies the UAP containing the object that detected the alert via its associated transport layer port.

**12.17.5.2.2.2.17 Alert source object**

The alert source object indicates the object instance that detected the alarm condition.

NOTE The alert reporting management object reports alert conditions detected by one or more alert detecting objects.

**12.17.5.2.2.2.18 Detection time**

This parameter specifies the time at which the alert condition was detected. This value indicates the network time at which the alert was detected. How time information is made available to an application reporting an alert is a device local matter, not specified by this standard.

NOTE Translating network time to social time (wall clock time), if needed, is performed in the gateway. See 5.6 for further details.

**12.17.5.2.2.2.19 Alert class**

This parameter indicates if this is an event (stateless) or alarm (state-oriented) type of alert.

#### **12.17.5.2.2.20 Alarm direction**

For alerts that are state-oriented (alarms), this indicates if the report is for an alarm condition, or a return to normal from an alarm condition.

#### **12.17.5.2.2.21 Alert category**

Alert category indicates if the alert is a device diagnostic alert, a communication diagnostic alert, a security alert, or a process alert.

#### **12.17.5.2.2.22 Alert priority**

Alert priority is a value that suggests the importance of the alert. A larger value implies a more important alert. Host systems map device priorities into host alert priorities that typically include urgent, high, medium, low, and journal. The recommended mapping of alert priority values into these categories is as follows:

- 0 – 2: journal
- 3 – 5: low
- 6 – 8: medium
- 9 – 11: high
- 12 – 15: urgent

#### **12.17.5.2.2.23 Alert type**

Alert type provides additional information regarding the alert, specific to the alert category.

#### **12.17.5.2.2.24 Alert value size**

Alert value size specifies the number of octets of the alert value.

#### **12.17.5.2.2.25 Alert value**

The alert value is present if and only if alert value size is greater than zero.

### **12.17.5.2.3 AlertAcknowledge service**

#### **12.17.5.2.3.1 General**

AlertAcknowledge is a two-part service that is used to acknowledge an individual alert to an alert reporting management object. For unicast alert reports, receipt of an AlertAcknowledge shall result in the ceasing of AlertReport retry requests for the corresponding individual alert.

An AlertAcknowledge shall be sent for every AlertReport received.

**NOTE** If a duplicate AlertReport has been received, either the application that sent the AlertReport did not receive the AlertAcknowledge within its timeout/retry time, or the AlertAcknowledgement message was not received. Since the application sending the AlertAcknowledge cannot be certain which situation occurred, a duplicate acknowledgement is sent.

The AlertAcknowledge service is described in Table 279.

**Table 279 – AlertAcknowledge service**

Parameter name	Request	Indication
Argument	M	M
Service contract identifier	M	
Priority	M	
Discard eligible	M	
End-to-end transmission time		M
Source network address		M
Source TDSAP	M	
Source transport layer port		M
Source object identifier	M	M(=)
Destination transport port	M	
Destination TDSAP		M
Destination object identifier	M	M(=)
Individual alert identifier	M	M(=)

### **12.17.5.2.3.2 Argument– service contract identifier**

#### **12.17.5.2.3.2.1 General**

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

#### **12.17.5.2.3.2.2 Priority**

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

#### **12.17.5.2.3.2.3 Discard eligible**

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

NOTE This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

#### **12.17.5.2.3.2.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

#### **12.17.5.2.3.2.5 Source network address**

This parameter identifies the network address for the source of this request.

#### **12.17.5.2.3.2.6 Source transport layer data service access point**

This parameter identifies the service primitive source APs associated TDSAP. The TDSAP maps 1-to-1 to a UAP.

#### **12.17.5.2.3.2.7 Source transport layer port**

This parameter identifies the transmitting application source UAP associated with the transport layer port.

#### **12.17.5.2.3.2.8 Source object identifier**

This parameter identifies the object that is initiating the alert acknowledgement.

#### **12.17.5.2.3.2.9 Destination transport layer port**

This parameter identifies the application process to receive the alert acknowledgement as a single application is associated with a TL port.

#### **12.17.5.2.3.2.10 Destination transport layer data service access point**

This parameter identifies the TDLE SAP corresponding to the destination transport port.

#### **12.17.5.2.3.2.11 Destination object identifier**

This parameter identifies the object in the application process of the device to receive the acknowledgement.

#### **12.17.5.2.3.2.12 Individual alert identifier**

This parameter identifies the individual alert that is being acknowledged.

### **12.17.6 Client/server and source sink commonalities**

#### **12.17.6.1 Individual or concatenated messaging for client/server and/or source/sink**

Client/server and source/sink messages may be sent as an individual transport service data unit (TSDU), or may be concatenated together within a single TSDU. Concatenation supports both four-part primitive messages (requests and responses) and two-part primitive messages (requests only). Concatenation allows client/server messages to be combined in the TSDU with source/sink messages. How APDU concatenation is determined and accomplished is a device local matter and is not specified by this standard. It is recommended that concatenations refrain from including more than two services, as this may result in more bursty communication.

NOTE 1 The reader may wish to reference the discussion of stretch ACK violation in IETF RFC 2525 for a discussion of message acknowledgement concatenation and the ramifications of having more than two such acknowledgements in a concatenation.

NOTE 2 Publish/subscribe already supports including multiple values from a UAP in a single message. See 12.15.2.5 and 12.15.2.6 for further details.

Concatenation may be used to reduce transmission overhead and/or deliver a set of messages to the corresponding ASL as a unit.

All messages within the concatenation shall have a common message priority, and shall indicate communication via a common communication contract.

The number of ASL services that may be concatenated by an ASL building the concatenated PDU is limited by the maximum APDU size corresponding to the communication contract to be used for the messages.

The ASL receiving a concatenated APDU is required to parse and handle each APDU individually from start to finish until the end of the TSDU has been met. If a protocol error is detected during parsing of a concatenated APDU, a single malformed APDU error is indicated and the remaining portion of the APDU shall be discarded.

An ASL concatenation may contain:

- Homogeneous ASL service primitives (e.g., all service requests); or
- Heterogeneous ASL service primitives (e.g., for client/server flows, requests and responses; may be mixed).

An ASL concatenation may contain:

- Homogeneous application communication flow primitives (e.g., all client/server); or
- Heterogeneous application communication flow type primitives (e.g., client/server and source/sink)

The AL itself makes no requirements on how responses to services included in a concatenation are returned; determination of such is at the discretion of the receiving application. For example, if a concatenated client/server request contains service requests (A, B, C) and another concatenated client/server service request contains service requests (D, E), the client/server responses for these may:

- Not be required (e.g., A, B, and D may be four-part services that require a response, but C and E may be two-part services that do not require a response);

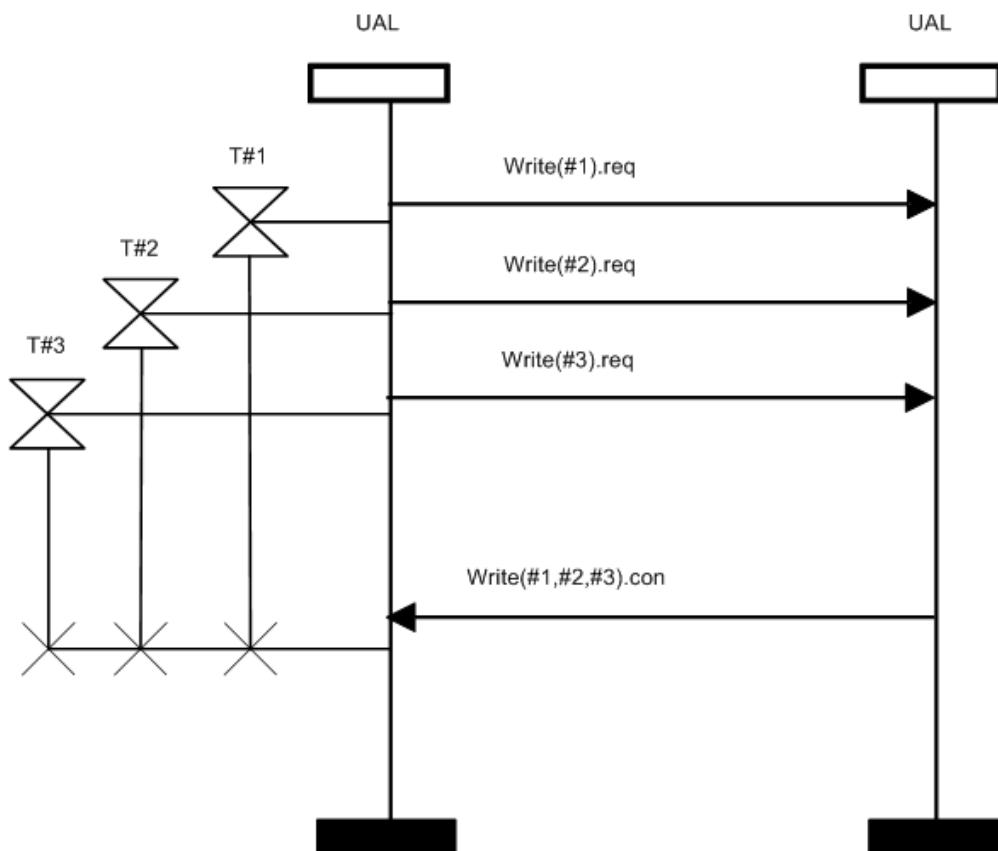
- Not be concatenated at all, and returned in any order (e.g., response A, response B, response E, response D, response C, all in separate APDUs);
- Be partially concatenated, in any order (e.g., response returned may be B,C in one APDU, A in another APDU);
- Employ a single APDU to respond to a concatenated request, but responses may be concatenated in any order (e.g., response returned may be concatenated as BCA);
- Be fully concatenated in the same order (e.g., response returned concatenated as ABC);
- Be entirely differently concatenated than the requests received (e.g., response may be returned BD, ACE).

How and when an ASL initiating a communication determines when to create a concatenation as well as when to deliver the concatenation to the lower communication protocol suite for conveyance; this is a local matter, and is not specified by this standard. However, this standard shall specify the overall structure of a TSDU containing concatenated APDUs.

**NOTE 3** This standard does not prescribe the order in which services included in a concatenate are handled by the destination. For example, the order of responses may differ from the order of the requests included in the concatenation.

**NOTE 4** An implementation may choose to have further control over concatenation content by defining local services to bracket concatenated constructions.

Figure 133 illustrates a concatenated response for multiple outstanding write requests with no message loss. Timeouts are described in 12.12.4.2.2.1.



**Figure 133 – Concatenated response for multiple outstanding write requests (no message loss)**

#### 12.17.6.2 Application sub-layer common services for client/server and source/sink messaging – Tunnel

##### 12.17.6.2.1 General

The tunnel service may use either a two-part (source/sink) or a four-part (client/server) communication service primitive model.

Information to enable matching service primitives, effecting appropriate application level handling of error situations such as duplicate message detection, detection of out-of-order delivery, etc. shall be the responsibility of the application process to include within the non-native payload of the tunnel messages.

The tunnel service can be used to encapsulate both client/server and source/sink communications as defined by this standard. This service identifies a message as destined for a non-native protocol tunnel. The service identifier is required so that the ASL can parse a message and determine whether to pass it on to a legacy protocol tunnel or handle it as a native message. The tunnel service provides a single level of message encapsulation for a protocol tunnel. The non-native APDU is passed through to the destination object specified in the tunnel service request.

NOTE A tunneling application may establish the retry policy for two-part (source/sink) tunnel requests that it sends.

Table 280 defines the tunnel service primitives.

**Table 280 – Tunnel service**

Parameter name	Request	Indication	Response	Confirm
Argument	M	M		
Service contract identifier	M			
Priority	M			
Discard eligible	M			
End-to-end transmission time		M		
Forward congestion notification		M		
Application destination transport layer port	M			
Application destination TDLE SAP		M		
Application destination object identifier	M	M(=)		
Application source network address		M		
Application source TLDE SAP	M			
Application source transport layer port		M		
Application source object identifier	M	M(=)		
Payload length (in octets)	M	M(=)		
Tunnel payload data	M	M(=)		
Result			U	U
Service contract identifier			M	
Priority			M	
Discard eligible			M	
End-to-end transmission time				M
Forward congestion notification				M
Forward congestion notification echo			M	M(=)
Application destination transport layer port			M	
Application destination TDLE SAP				M
Application destination object identifier			M	M(=)
Application source network address				M
Application source TLDE SAP			M	
Application source transport layer port				M(=)
Application source object identifier			M	M(=)
Payload length (in octets)			M	M(=)
Tunnel payload data			M	M(=)

### 12.17.6.2.2 Argument

#### 12.17.6.2.2.1 Service contract identifier

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

### **12.17.6.2.2.2 Priority**

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

### **12.17.6.2.2.3 Discard eligible**

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

**NOTE** This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

### **12.17.6.2.2.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

### **12.17.6.2.2.5 Forward congestion notification**

This parameter indicates if the request has encountered network congestion on its path from the client to the server.

### **12.17.6.2.2.6 Application destination transport layer port**

This parameter identifies the UAP at the destination for this service request.

### **12.17.6.2.2.7 Application destination transport layer data service access point**

This parameter represents the TDSAP corresponding to application layer transport destination port.

### **12.17.6.2.2.8 Application destination object identifier**

This parameter identifies the object in the receiving application.

### **12.17.6.2.2.9 Application source network address**

This parameter identifies the network address for the client of this request.

### **12.17.6.2.2.10 Application source transport layer data service access point**

This parameter identifies the application source UAP's associated TDSAP. The TDSAP maps 1-to-1 to a UAP. The UAP contains the object originating the request in the client.

### **12.17.6.2.2.11 Application source transport layer port**

This parameter identifies the UAP that is the source for this service request.

### **12.17.6.2.2.12 Application source object identifier**

This parameter indicates the application source object that is originating the tunnel service request.

### **12.17.6.2.2.13 Payload length**

This parameter indicates the number of octets of the tunnel payload parameter.

### **12.17.6.2.2.14 Tunnel payload data**

This parameter represents the data (e.g., legacy protocol APDU) that is to be conveyed to the server object.

### **12.17.6.2.3 Response**

#### **12.17.6.2.3.1 Service contract identifier**

This parameter identifies the communication service contract agreement that was made between the UAP requesting the service and the local DMAP. The value shall be in the set of valid values for a contract identifier as defined by 6.3.11.

**12.17.6.2.3.2 Priority**

This parameter defines the message priority of service that is required of the communication. Possible values are high or low. Transmission and delivery of high-priority messages is more important than transmission and delivery of messages of low priority.

**12.17.6.2.3.3 Discard eligible**

This parameter defines the guidance to the communication network regarding the application impact of discarding the application message in the event of network congestion. Possible values are true (the message may be considered for discard), or false (do not consider the message for discard).

**NOTE** This guidance is provided for use by routers that are constructed to employ an intelligent message discard policy rather than a random discard policy in situations of network congestion.

**12.17.6.2.3.4 End-to-end transmission time**

This is the transmission time from the transport layer at the requesting device to the transport layer at the receiving device.

**12.17.6.2.3.5 Forward congestion notification**

This parameter indicates if the service response encountered network congestion on its path from the server to the client.

**12.17.6.2.3.6 Forward congestion notification echo**

This parameter indicates if the service request encountered network congestion on its path from the client to the server.

**12.17.6.2.3.7 Application destination transport layer port**

This parameter identifies the UAP at the destination for this service request.

**12.17.6.2.3.8 Application destination transport layer data service access point**

This parameter represents the TDSAP corresponding to application layer transport destination port.

**12.17.6.2.3.9 Application destination object identifier**

This parameter indicates the application destination object that is originating the tunnel service request.

**12.17.6.2.3.10 Application source network address**

This parameter identifies the network address for the client of this request.

**12.17.6.2.3.11 Application source transport layer data service access point**

This parameter identifies the application source UAP's associated TDSAP. The TDSAP maps 1-to-1 to a UAP. The UAP contains the object originating the request in the client.

**12.17.6.2.3.12 Application source transport layer port**

This parameter indicates the application source object that is originating the tunnel service request.

**12.17.6.2.3.13 Application source object identifier**

This parameter indicates the application source tunnel object that is originating the tunnel service request.

**12.17.6.2.3.14 Payload length**

This parameter indicates the number of octets of the tunnel payload parameter.

**12.17.6.2.3.15 Tunnel payload data**

This parameter represents the data (e.g., legacy protocol APDU) that is to be conveyed to the server object.

## 12.18 Application layer flow use to lower layer services

### 12.18.1 General

All types of messaging (e.g., publication, client/server, source/sink, bulk transfer) and all qualities of service may flow through the common TDSAP for the application process. Table 281 indicates the mapping of the AL flows to the TL services provided.

**Table 281 – Application flow characteristics**

Application flow type	Buffered or queued	Periodic (scheduled)	Order sensitive	Reliable	Unacknowledged	Message importance		
						High	Medium	Low
Periodic publish/subscribe	Buffered	✓	✓		✓	✓	✓	✓
Client/ server	Queued				✓	✓	✓	✓
Source/sink	Queued				✓	✓	(No use case identified)	✓

### 12.18.2 Application layer use of transport layer data service access points

The ASL communicates with the lower layers of the communication protocol suite via TDSAPs. The information communicated to the transport layer for TDSAP mapping is a subset of the information communicated to the ASL from the UAP at the ASAP. There is one well-known TDSAP in this standard, TDSAP number 0, that is used for communications with the objects represented by the DMAP application.

TDSAPs that are not well known may be associated with one and only one particular application process. That is, there is a one-to-one mapping relationship between a TDSAP and a UAL process. Because TDSAPs are local, remote entities indicate a transport layer port that represents a corresponding application. Transport layer ports map 1-to-1 to TDSAPs. UAL process / TDSAP / transport data port relationships shall survive application restart.

NOTE Fifteen TDSAPs are available for compressed transmission. See 11.6 for further details.

### 12.18.3 Mapping application services primitives to transport layer service primitives

Table 282 indicates the mapping of application service primitives to transport services.

**Table 282 – Application service primitive to transport service primitive mapping**

Application service primitive	Transport service	Data conveyed between application sub-layer and transport service
Publish.request Read.request, Read.response Write.request, Write.response Execute.request, Execute.response Tunnel.request, Tunnel.response AlertReport.request AlertAcknowledge.request	T-DATA.request	Contract_ID APDU_size APDU Message priority Discard eligibility Source TDSAP Destination transport layer port  NOTE 1 The contract ID indicates destination address, and contract priority. NOTE 2 The source transport layer port can be determined from the source TDSAP, but is explicitly passed to match the interface provided by the transport layer.
Publish.indicate Read.indicate, Read.confirmation Write.indicate, Write.confirmation Execute.indicate, Execute.confirmation Tunnel.indicate, Tunnel.confirmation AlertReport.indicate AlertAcknowledge.indicate	T-DATA.indicate	128-bit source network address Source transport layer port APDU_size (equivalent to TSDU size) APDU (equivalent to TSDU) Explicit congestion notification (ECN) Destination TDSAP Destination transport layer port Transport time (one-way end-to-end delivery time, in seconds)

## 12.19 Application layer management

### 12.19.1 General

Application layer management supports the local DMAP application sub-layer management object. Access to the attributes and methods of this object are defined by the ASL. For this standard, the ASL provides access to read a configured value from the ASL-MIB, to write a configured value to the ASL-MIB, and to support reset of the ASL.

### 12.19.2 Application sub-layer handling of malformed application protocol data units

The ASL supports informing the local DMAP of a potential device/communication problem if an AL management-configured threshold is reached within a configured time period, for malformed APDUs received from a particular source device. Some examples of malformed APDUs are:

- APDU length incorrect (too long or too short);
- Invalid service identifier; or
- Service misuse (e.g., response primitive was indicated in the PDU for a two-part client/server or source/sink service).

The intent of this information is to enable the DMAP to provide information to higher level management. This may be important, for example, to enable detection of a malformed APDU attack occurs.

The ASL may be configured to advise the DMAP whenever a malformed APDU is received.

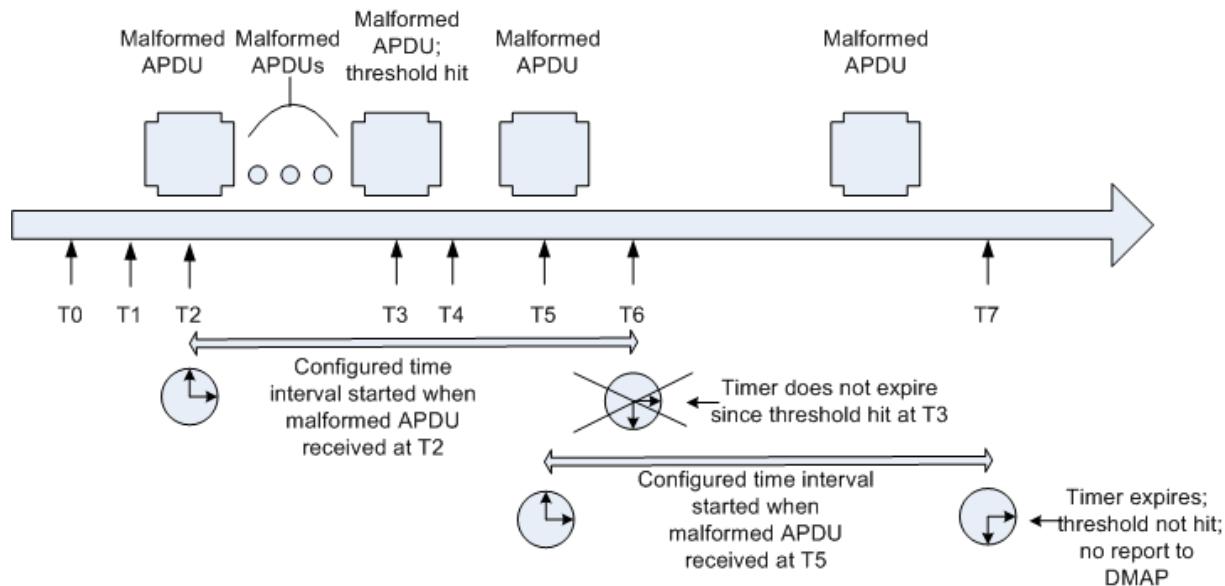
The ASL may be configured with non-zero values for a threshold and a time interval. When so configured, the ASL will maintain individual counters and timers internally for each network source address from which a malformed APDU has been received. Counting commences with the receipt of the first malformed APDU from a device and continues until either the malformed APDU threshold value is reached or the ASL\_TimePeriodForMalformedAPDUs expires.

If the malformed APDU threshold value is reached prior to or at the expiration of the configured time interval, the DMAP is advised and the count and time interval for the device are reset.

If the malformed APDU threshold is not reached within the configured time interval, the counters and timers are internally reset.

NOTE How the DMAP is advised of a malformed APDU or that a threshold has reached in a time interval is a local matter, and hence is not specified by this standard.

Figure 134 illustrates the handling of malformed APDUs.



**Figure 134 – Management and handling of malformed APDUs received from device X**

### 12.19.3 Application sub-layer management object attributes

Table 283 describes the attributes supported by the application sub-layer management object (ASLMO).

**Table 283 – ASLMO attributes**

<b>Standard object type name: Application sub-layer management object</b>				
<b>Standard object type identifier: 121</b>				
<b>Defining organization: ISA 100.11a</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Accessibility: N/A Initial default value: 7 Valid value set: 7	N/A
Reserved for future use	0			
MalformedAPDUsAdvise	1	Indicates if ASL should indicate to local DMAP whenever a malformed APDU is encountered	Type: Boolean Classification: Static Accessibility: Read/write Initial default value : Disabled Valid value set: 0 = Disabled 1 = Enabled	If this parameter has a value of True, the ASL shall pass each malformed APDU it receives on to the local DMAP
TimeIntervalForCounting MalformedAPDUs	2	This attribute specifies the time interval for the ASL to count malformed APDUs received from a particular device. Counting occurs from detection of the first malformed APDU from a device. This interval is applied commonly to APDUs from all source network addresses. The units of this attribute is seconds	Type: TAITimeDifference Classification: Static Accessibility: Read/write Initial default value : 0 Valid value set: $0 \leq \text{value} \leq 86,400$ seconds Number of days is not included (it is always equal to zero)	If the time interval expires without the threshold being met, the corresponding internal malformed ASL counter and timer information shall be reset to zero (0)
MalformedAPDUsThreshold	3	Common threshold value to apply to malformed APDUs received from each device	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value : 0 Valid value set: 0 to 65535	If this threshold is met in the specified time interval, a communication alert shall be reported indicating the device that has been sending malformed APDUs.  If a threshold value is set while counting is in progress, and the value set is lower than the prior threshold such that the new threshold has been exceeded, a malformed APDU alert shall be reported
MalformedAPDUAAlertDescriptor	4	Describes how the malformed alert is reported on the network	Type : Alert report descriptor Classification: Static Accessibility: Read/write Initial default value: Alert report disabled = True Alert report priority = 7	

<b>Standard object type name: Application sub-layer management object</b>				
<b>Standard object type identifier: 121</b>				
<b>Defining organization: ISA 100.11a</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
			Valid value set: See type definition	
MaxDevicesForWhichMalformedAPDUsCanBeCounted	5	Describes the malformed APDU counting capability of the ASL in terms of the maximum number of devices for which counts can be simultaneously maintained	Type: Unsigned16 Classification: Constant Accessibility: Read only Initial default value: N/A Valid value set: 0 to 65535	A minimum value required may be established for example based on the role of the device
Reserved for future use by this standard	6-63			

Attributes classified as either constant or static shall be preserved across restarts and power-fails.

#### **12.19.4 Application sub-layer management object methods**

##### **12.19.4.1.1 Standard object methods**

An ASLMO has the methods defined in Table 284.

**Table 284 – Application sub-layer management object methods**

<b>Standard object type name: Application sub-layer management object</b>		
<b>Standard object type identifier: 121</b>		
<b>Defining organization: ISA</b>		
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>
Null	0	Reserved by standard for future use
Reset	1	Reset application sub-layer
Reserved for future use by this standard	2-127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

##### **12.19.4.1.2 Reset method**

Table 285 specifies the reset method primitives.

**Table 285 –Reset method**

<b>Standard object type name: Application sub-layer management object (ASLMO)</b>			
<b>Standard object type identifier: 121</b>			
<b>Defining organization: ISA 100.11a</b>			
Method name	Method ID	Method description	
Reset	1	Reset application sub-layer	
<b>Input arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
1	ResetType	Type: Unsigned8 Valid value set: 0 – not used 1 – reset to factory default settings 2 – reset to provisioned settings 3 – warm reset (reset to provisioned settings and any communication contract related information) 4 – reset all dynamic data (e.g., related to statistics) Values 5 to 255 are reserved for future use by this standard	Type of reset desired
<b>Output arguments</b>			
Argument #	Argument name	Argument type (data type and length)	Argument description
None			

#### 12.19.4.2 Input arguments

The ResetType parameter indicates the type of reset desired. The sub-layer may be reset to:

- Factory default settings;
- Only maintain provisioned settings (if any);
- Only maintain the set of both provisioned settings and communication contract settings (if any); or
- Reset all dynamic statistics.

NOTE An example of default factory settings may, for example, be:

- MalformedAPDUsAdvise configuration parameter, indicating disabled;
- TimeIntervalForCountingMalformedAPDUs configuration parameter, indicating 100 APDUs; and
- MalformedAPDUsThreshold configuration parameter, indicating a zero time interval.

#### 12.19.4.3 Output arguments

There are no output arguments for this method.

#### 12.19.4.4 Response codes

The following feedback codes are valid for this method:

- success;
- other;
- invalidArgument; and
- vendorDefined.

#### 12.19.5 Application sub-layer management object alerts

Table 286 defines the alerts for the ALSMO.

**Table 286 – ASLMO alerts**

<b>Standard object type name(s): Application sub-layer management object</b>					
<b>Standard object type identifier: 121</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Malformed APDU alert</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority	Value data type	Description of value included with alert
Event	Communication diagnostic	malformedAPDU CommunicationAlert	7 (this value is a mid- range medium priority alert)	Type: No special standard type is defined as the protocol content does not correspond to an attribute of the ASLMO. Rather, it is constructed within protocol as an implicit sequence, the construction of which is identifiable by the combination of alert class, alert category and alert type.	Three elements are included in the following sequence: 1. Source address of malformed APDUS (128 bit address) 2. Threshold value exceeded (Unsigned16) 3. Time interval in which threshold was exceeded (TAITimeDifference)
				Initial default value: N/A	
				Valid value set: N/A	

### **12.19.6 Device management application process services invoked by application sub-layer**

If the configured ASL malformed APDUs interval and ASL malformed APDUs threshold are both non-zero, the ASL shall commence keeping malformed APDU statistics. If the threshold is reached prior to or upon expiration of the configured time interval, the ASL shall report to the DMAP that a communication diagnostic alert should be generated. The data provided by ASL to the DMAP shall include:

- An indication that malformedAPDUsThresholdReached situation has been reached.

NOTE This is indicated if and when the ASL malformed APDUs threshold is reached for malformed APDUs received from a single source network address within the configured ASL malformed APDUs interval.

- Diagnostic information regarding the malformed APDUs detected, such as:
- Number of APDUs received that did not have correct length;
- Number of APDUs received with an invalid service identifier; and
- Number of APDUs received with service identifier misused.
- The 128-bit source network address of the malformed APDUs;
- The threshold value that was reached; and
- The time duration over which the malformed APDUs were received. This time interval is calculated as the time from detection of the first malformed APDU from the indicated device by the ASL and the detection of the malformed APDU that resulted in the ASL threshold limit being reached for the indicated device. This time duration shall be less than or equal to the configured time interval established in the ASL management parameters.

### **12.19.7 Process industries standard objects**

#### **12.19.7.1 General**

The standard objects defined in this standard are included in this subclause. These objects address the fundamental needs of the process industry. The unified field theory (for process control) that underlies this standard defines standard field objects leveraging existing field

device object definitions from field-proven object-oriented process control system standards. The set of objects has been selected based on commonality of use and are defined to enable interoperability among devices.

NOTE This standard presumes access restrictions to object attributes, if necessary to satisfy system usage requirements, are enforced by human interface devices and/or gateway devices.

To support timestamps used in process control industry alarm reports, the time value construct used to represent time shall support a coding accuracy of 1 ms. This accuracy is necessary when supporting high speed resolution typical of process industry sequence of event (SOE) applications.

#### **12.19.7.2 Process industries user application objects**

A basic list of user application objects is anticipated for the process control industries profile. The unified field objects (UFOs) defined in this standard are:

- Analog input object;
- Analog output object;
- Binary input object; and
- Binary output object.

Application object control mode supports the following modes:

- Target mode is the mode to which device was commanded to transition. This may be different from actual mode if the device cannot accept the target mode due to error, etc.
- Actual mode is the current mode of the object.
- Normal mode is the operating mode of the block that is desired by the responsible control engineer. Normal mode is one of the modes other than OOS designated as “normal operation” for the block.
- Permitted modes represent the set of modes that are valid for this object. This is a filter that can be applied to limit the target mode enumeration. For example, manual mode may be disabled this way. OOS is always included in the set of permitted modes.

Supported modes include:

- Out of service (OOS): Device is not actively measuring PV value or not accepting OP value. Another common name for this mode is “inactive”. The value and associated status indicating OOS condition is still communicated by the device. This is not intended to disable communication;
- Manual (MAN): Process value can be manually entered by the operator, used for open loop control with human in the loop or for overriding faulty measurement. Also useful for testing; and
- Automatic (AUTO): Device is actively measuring PV value or accepting OP value.

#### **12.19.7.3 Analog input user object**

##### **12.19.7.3.1 General**

A standard analog input user object representing an encapsulation of an analog input is defined. If multiple analog inputs are represented by a device, multiple analog input user objects should be instantiated. Object type-specific attributes of this object include:

- Process value: a floating point value represented in engineering units and status;
- Mode: a structured attribute representing target mode, actual mode, permitted mode, and normal mode;
- Corresponding concentrator object: specifies the concentrator associated to publish the PV; and
- Scale: represents the range and units of the process value via a structured attribute that indicates 100% value range, 0% value range, coded representation of engineering units, and significant digits for display.

Standard alerts for this object also will be defined.

NOTE 1 A structured attribute is required to be added for each type of alert report supported by the object. This attribute supports enabling/disabling of an alert report and establishes the alert priorities.

NOTE 2 For alarms, a structured attribute may additionally be required to establish alarm limits, to indicate if an alarm is present.

#### **12.19.7.3.2 Object attributes**

An analog input object has the attributes defined in Table 287.

**Table 287 – Analog input object attributes**

<b>Standard object type name: Analog input object</b>				
<b>Standard object type identifier: 99</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Accessibility: N/A Initial default value: N/A Valid value set: 1 to 65535	N/A
Reserved for future use	0			
PV	1	Measurement variable in engineering units of the sensor	Type: Process control value and status for analog value Classification: Dynamic Accessibility: Read only Initial default value: NaN Status: Unknown; Substatus: Unknown Limit condition: Not limited  Valid value set: See definition of process control value and status for analog value structure	Analog process value and status of that value. Accessibility is read/write only when MODE.Target=MAN See 12.19.7.2. When a write to PV is done, the device may implement this as a write to a non-network visible internal variable, and use the non-visible value when constructing the value it represents for the PV. As appropriate, the device may report a different status for the PV than that which was provided in the write request
Mode	2	Mode	Type: Process control mode Classification: Static Accessibility: Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access Initial default value: Actual mode value indicates OOS Valid value set: See Process control mode structure type definition	Actual, target, permitted, and normal mode values
Reserved for future use	3			
Scale	4	Range and units of the measurement variable	Type: Process control scaling data Classification: Static Accessibility: Read/write Initial default value: Engineering units values for 0% and for 100% BOTH indicate 0 Valid value set: See definition of scale structure type	Scaling information for the analog process value
Reserved for future use by this standard	5-25			N octets of presently undefined content

**12.19.7.3.3 Standard object methods**

An analog input object has the methods defined in Table 288.

**Table 288 – Analog input object methods**

<b>Standard object type name: Analog input object</b>		
<b>Standard object type identifier: 99</b>		
<b>Defining organization: ISA</b>		
Method name	Method ID	Method description
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0-127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

#### **12.19.7.3.4 Alerts**

An analog input may report the alerts shown in Table 289. If an alert is supported, a corresponding alert descriptor attribute shall be added to the analog input object to describe the characteristics of the alert.

**Table 289 – Analog input alerts**

<b>Standard object type name(s):Analog input</b>					
<b>Standard object type identifier: 99</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Analog input alerts</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type(s) (Enumerated: based on alert category)	Alert priority	Value data type and size	Description of value included with alert
Alarm	Process	1	High	Any	Type: Float32 Initial default value: N/A Valid value set: As defined by IEEE for Float32
Alarm	Process	2	HighHigh	Any	Type: Float32 Initial default value: N/A Valid value set: As defined by IEEE for Float32
Alarm	Process	3	Low	Any	Type: Float32 Initial default value: N/A Valid value set: As defined by IEEE for Float32
Alarm	Process	4	LowLow	Any	Type: Float32 Initial default value: N/A Valid value set: As defined by IEEE for Float32
Alarm	Process	5	DeviationLow	Any	Type: Float32 Initial default value: N/A Valid value set: As defined by IEEE for Float32
Alarm	Process	6	DeviationHigh	Any	Type: Float32 Initial default value: N/A Valid value set: As defined by IEEE for Float32
Alarm	Process	0	OutOfService	Any	Type: Float32 Initial default value: N/A Valid value set: As defined by IEEE for Float32

#### 12.19.7.4 Analog output user object

##### 12.19.7.4.1 General

A standard analog output user object represents an encapsulation of an analog output. If multiple analog outputs are represented by a device, multiple analog output user objects should be instantiated. Object type-specific attributes of this object include:

- Commanded output value: a floating point value represented in engineering units and status;

- Mode: a structured attribute representing target mode, actual mode, permitted mode, and normal mode;
- Readback: value and status of the actual position;
- Provider of OP value: indicates the source of the OP value
- Corresponding concentrator object: specifies the concentrator associated to publish the Readback value; and
- Scale, representing range and units of the process value via a structured attribute that indicates 100% value range, 0% value range.

NOTE 1 A structured attribute is required to be added for each type of alert report supported by the object. This attribute supports enabling/disabling of an alert report and establishes the alert priorities.

NOTE 2 For alarms, a structured attribute may additionally be required to establish alarm limits, to indicate if an alarm is present.

#### **12.19.7.4.2 Object attributes**

An analog output object has the attributes defined in Table 290.

**Table 290 – Analog output attributes**

<b>Standard object type name: Analog output object</b>				
<b>Standard object type identifier: 98</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16	N/A
			Classification: Constant	
			Accessibility: N/A	
			Initial default value: N/A	
			Valid value set: 1 to 65535	
Reserved for future use	0			
OP	1	Output value for the actuator	Type: Process control value and status for analog value structure	This is the standard attribute that serves as the destination attribute for a publication from another object.
			Classification: Dynamic	
			Accessibility: Read/write	
			Initial default value: NaN Status: Unknown; Substatus: Unknown Limit condition: Not limited	
			Valid value set: See definition of process control value and status for analog value structure	
Mode	2	Mode	Type: Process control mode	Actual, target, permitted, and normal mode values
			Classification: Static	
			Accessibility: Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access	
			Initial default value: Actual mode value indicates OOS	
			Valid value set: See Process control mode structure type definition	
Reserved for future use	3			
Readback	4	Readback value – the actual position of the OP	Type: Process control value and status for analog value	Analog process value and status of that value  This is the standard attribute that is published from this object.
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: NaN; Status: Unknown; Substatus: Unknown Limit condition: Not limited	
			Valid value set: See definition of process control value and status for analog value structure	
Reserved for future use	5			
Scale	6	Range and units of the output variable	Type: Process control scaling data structure	Scaling information
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Engineering units values for 0% and for 100% BOTH indicate 0	
			Valid value set: See definition of Scale structure type	

<b>Standard object type name: Analog output object</b>				
<b>Standard object type identifier: 98</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Reserved for future use by this standard	7-25			N octets of presently undefined content

#### 12.19.7.4.3 Standard object methods

An analog output object has the methods defined in Table 291.

**Table 291 – Analog output object methods**

<b>Standard object type name: Analog output object</b>		
<b>Standard object type identifier: 98</b>		
<b>Defining organization: ISA</b>		
Method name	Method ID	Method description
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0-127	These method identifiers are reserved for future use by this standard.
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

#### 12.19.7.4.4 Alerts

An analog output may report the alerts shown in Table 292. If an alert is supported, a corresponding alert descriptor attribute shall be added to the analog output object to describe the characteristics of the alert.

**Table 292 – Analog output alerts**

<b>Standard object type name(s): Analog output</b>						
<b>Standard object type identifier: 98</b>						
<b>Defining organization: ISA</b>						
<b>Description of the alert: Analog output alerts</b>						
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type(s) (Enumerated: based on alert category)		Alert priority	Value data type and size	Description of value included with alert
Alarm	Process	1	High	Any	Type: Float32	Process variable
					Initial default value: N/A	
					Valid value set: As defined by IEEE for Float32	
Alarm	Process	2	HighHigh	Any	Type: Float32	Process variable
					Initial default value: N/A	
					Valid value set: As defined by IEEE for Float32	
Alarm	Process	3	Low	Any	Type: Float32	Process variable
					Initial default value: N/A	
					Valid value set: As defined by IEEE for Float32	
Alarm	Process	4	LowLow	Any	Type: Float32	Process variable
					Initial default value: N/A	
					Valid value set: As defined by IEEE for Float32	
Alarm	Process	5	DeviationLow	Any	Type: Float32	Process variable
					Initial default value: N/A	
					Valid value set: As defined by IEEE for Float32	
Alarm	Process	6	DeviationHigh	Any	Type: Float32	Process variable
					Initial default value: N/A	
					Valid value set: As defined by IEEE for Float32	
Alarm	Process	0	OutOfService	Any	Type: Float32	Process variable
					Initial default value: N/A	
					Valid value set: As defined by IEEE for Float32	

### **12.19.7.5 Binary input user object**

#### **12.19.7.5.1 General**

A standard binary input user object represents an encapsulation of a single binary input. If multiple binary inputs are represented by a device, multiple binary input user objects should exist to represent them. Object type specific attributes of this object include:

- Process value: binary value and status;
- Mode: a structured attribute representing target mode, actual mode, permitted mode, and normal mode; and
- Corresponding concentrator object: specifies the concentrator associated to publish the process value.

NOTE A structured attribute is required to be added for each type of alert or alarm report supported by the object. This attribute supports enabling/disabling of an alert report and establishes the alert priorities.

#### **12.19.7.5.2 Object attributes**

A binary input object has the attributes defined in Table 293.

**Table 293 – Binary input object attributes**

<b>Standard object type name: Binary input object</b>				
<b>Standard object type identifier: 97</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Accessibility: N/A Initial default value: N/A Valid value set: 1 to 65535	N/A
Reserved for future use	0			
PV_B	1	Discrete measurement variable	Type: Process control value and status for discrete value Classification: Dynamic Accessibility: Read only Initial default value: 0 Status: Unknown; Substatus: Unknown Limit condition: Not limited  Valid value set: See definition of process control value and status for discrete value structure	Binary process value and status of that value  This is the standard attribute that is published from this object.  If this object is extended, such that another attribute needs to be published, a concentrator object(s) represents the resulting publication(s).  Accessibility is read/write only when in MAN mode. See 12.19.7.2. When a write to PV_B is done, the device may implement this as a write to a non-network visible internal variable, and use the non-visible value when constructing the value it represents for the PV_B. As appropriate, the device may report a different status for the PV_B than that which was provided in the write request.
Mode	2	Mode	Type: Process control mode Classification: Static Accessibility: Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access Initial default value: Actual mode value indicates OOS Valid value set: See Process control mode structure type definition	Actual, target, permitted, and normal mode values
Reserved for future use by this standard	3-25			N octets of presently undefined content

### 12.19.7.5.3 Standard object methods

A binary input object has the methods defined in Table 294.

**Table 294 – Binary input object methods**

<b>Standard object type name: Binary input object</b>		
<b>Standard object type identifier: 97</b>		
<b>Defining organization: ISA</b>		
Method name	Method ID	Method description
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0-127	These method identifiers are reserved for future use by this standard
Implementation-specific	128-255	These method identifiers are available for implementation-specific use

#### 12.19.7.5.4 Alerts

An analog output may report the alerts shown in Table 295. If an alert is supported, a corresponding alert descriptor attribute shall be added to the binary input object to describe the characteristics of the alert.

**Table 295 – Binary input alerts**

<b>Standard object type name(s): Binary input object</b>					
<b>Standard object type identifier: 97</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Binary input alerts</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type(s) (Enumerated: based on alert category)	Alert priority	Value data type and size	Description of value included with alert
Alarm	Process	1	DiscreteAlarm	Any	Type: Boolean Initial default value: N/A Valid value set: As defined by Boolean type definition
Alarm	Process	0	OutOfService	Any	Type: Boolean Initial default value: N/A Valid value set: As defined by Boolean type definition

#### 12.19.7.6 Binary output user object

##### 12.19.7.6.1 General

A standard binary output user object represents an encapsulation of a single binary output. If multiple binary outputs are represented by a device, multiple binary output user objects should exist to represent them. Object type specific attributes of this object include:

- Commanded output value: binary value and status;
- Mode: a structured attribute representing target mode, actual mode, permitted mode, and normal mode;
- Provider of OP\_B Value: indicates the source of the OP\_B value;
- Readback value: binary value and status; and
- Corresponding concentrator object: specifies the concentrator associated to publish the Readback\_B value.

NOTE A structured attribute is required to be added for each type of alert or alarm report supported by the object. This attribute supports enabling/disabling of an alert report and establishes the alert priorities.

##### 12.19.7.6.2 Object attributes

A binary output object has the attributes defined in Table 296.

**Table 296 – Binary output attributes**

<b>Standard object type name: Binary output object</b>				
<b>Standard object type identifier: 96</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
ObjectIdentifier	Object key identifier	Unique identifier for the object	Type: Unsigned16 Classification: Constant Accessibility: N/A Initial default value: N/A Valid value set: 1 to 65535	N/A
Reserved for future use	0			
OP_B	1	Discrete measurement variable	Type: Process control value and status for discrete value structure Classification: Dynamic Accessibility: Read only Initial default value: 0 Status: Unknown; Substatus: Unknown Limit condition: Not limited  Valid value set: See definition of process control value and status for analog value structure	This is the standard attribute that is the destination for a publication from another object.
Mode	2	Mode	Type: Process control mode Classification: Static Accessibility: Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access Initial default value: Actual mode value indicates OOS  Valid value set: See Process control mode structure type definition	Actual, target, permitted, and normal mode values
Reserved for future use	3			
Readback_B	4	Readback value of actual position of the actuator	Type: Process control value and status for discrete value Classification: Dynamic Accessibility: Read only Initial default value: 0; Status: Unknown; Substatus: Unknown Limit condition: Not limited  Valid value set: See definition of process control value and status for analog value structure	Analog process value and status of that value  This is the standard attribute that is published from this object.  If this object is extended, such that another attribute needs to be published, a concentrator object(s) represents the resulting publication(s)
Reserved for future use by this standard	5-25			N octets of presently undefined content

#### 12.19.7.6.3 Standard object methods

A binary output object has the methods defined in Table 297.

**Table 297 – Binary output object methods**

<b>Standard object type name: Binary output object</b>		
<b>Standard object type identifier: 96</b>		
<b>Defining organization: ISA</b>		
Method name	Method ID	Method description
Null	0	Reserved by standard for future use
Reserved for future use by this standard	0-127	These method identifiers are reserved for future use by this standard
Implementation-specific use	128-255	These method identifiers are available for implementation-specific use

#### 12.19.7.6.4 Alerts

A binary output may report the alerts shown in Table 298. If an alert is supported, a corresponding alert descriptor attribute shall be added to the binary output object to describe the characteristics of the alert.

**Table 298 – Binary output alerts**

<b>Standard object type name(s): Binary output object</b>					
<b>Standard object type identifier: 96</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Binary output alerts</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type(s) (Enumerated: based on alert category)	Alert priority	Value data type and size	Description of value included with alert
Alarm	Process	1	DiscreteAlarm	Any	Type: Boolean Initial default value: N/A Valid value set: As defined by Boolean type definition
Alarm	Process	0	OutOfService	Any	Type: Boolean Initial default value: N/A Valid value set: As defined by Boolean type definition

#### 12.19.8 Factory automation industries profile

##### 12.19.8.1 General

Additional standard objects to support the needs of factory automation may be added in future releases of the standard. More detailed thought specific to factory automation is deferred to a later release of this standard standardization activity. Examples of objects that may be defined to meet the needs of factory automation may include:

- Binary input user object (e.g., a contact);
- Binary output user object (e.g., a coil);
- Analog input user object;
- Analog output user object;
- Register input user object (e.g., to map 16 bits of two-state information often found in PLC input registers);
- Register output user object (e.g., to map 16 bits of two-state information often found in PLC output registers);
- Multi-state input user object (e.g., to map 8 bits of state information for a valve with enumerated states such as opening, open, closing, closed, or to map a unidirectional motor with states such as off, starting, running, stopping); and

- Multi-state output user object (e.g., to map 8 bits of state output information for an output device with enumerated states).

Standard alerts for these objects may also be defined.

#### 12.19.8.2 Manufacturer specific objects

Vendors may also define vendor- or device-specific custom objects. An example of a vendor-specific object for process systems is an LCD (liquid crystal display) object, in which a string can be stored for display on an LCD.

### 12.20 Process control industry standard data structures

#### 12.20.1 General

Process control industry standard data structures used shall be the data structures conveyed by the protocol defined by this standard. Industry-independent standard data structures and process control industry data structures are both summarized in Annex L of this standard.

NOTE Vendor-specific data structure definitions are not supported.

#### 12.20.2 Status for analog information

The status for analog information is a packed fixed format octet containing the items shown in Table 299.

**Table 299 – Status octet**

Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
<b>Quality</b>		<reserved>		<b>Quality dependent substatus</b>		<b>Limit condition</b>	
0 = Bad (value should not be used)		This bit shall always be set to zero		0 = non-specific 1 = configuration error 2 = not connected 3 = device failure 4 = sensor failure 5 = no communication with LUV 6 = no communication no LUV 7 = out of service (value is not being computed) All other values are reserved		0 = Not limited 1 = Low limit 2 = High limit 3 = Constant (both high and low limited)	
1 = Uncertain (value of less than normal quality)				0 = non-specific 1 = last usable value 2 = substituted or manual entry 3 = initial value 4 = sensor conversion inaccurate 5 = range limits exceeded 6 = sub normal All other values are reserved			
2 = Good (quality of value is good, but an alarm condition may exist)				0 = no special conditions exist. All other values are reserved			
3 = Reserved				All values are reserved. Within this standard, this shall always be set to zero			

NOTE The definitions for status octet are a subset of those defined by the HART Communication Foundation, Foundation Fieldbus, and the OPC Foundation.

#### 12.20.3 Value and status for analog information

As status does not indicate substitution, network-initiated writes using this analog data type structures are not permitted by this standard.

The structure of analog information is shown in Table 300.

**Table 300 – Data type: Process value and status for analog value**

<b>Standard data type name: Process control value and status for analog value</b>		
<b>Standard data type code: 65</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
Status	1	Type: Unsigned8, representing a Status Bit String Size: 1 octet Classification: Dynamic Accessibility: May vary by use Default value: Not specified Valid value set : See Table 299 for bit string construction details
Value	2	Type: IEEE 754_Float32 Size: 4 octets Classification: Dynamic Accessibility : May vary by use Default value : See IEEE 754 specification Valid value set: See IEEE 754 specification

**12.20.4 Value and status for binary information**

As status does not indicate substitution, network-initiated writes to digital data type structures are not permitted.

The structure of digital information is shown in Table 301.

**Table 301 – Data type: Process value and status for binary value**

<b>Standard data type name: Process control value and status for binary value</b>		
<b>Standard data type code: 66</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
Status	1	Type: Unsigned8, representing a Status Bit String Size: 1 octet Classification: Dynamic Accessibility: May vary by use Default value: Not specified Valid value set: 0 to 255; see Table 299 for bit string construction details
Value	2	Type: Boolean Size: 1 octet Classification: Dynamic Accessibility: May vary by use Default value: Not specified Valid value set: 0 or non-zero

**12.20.5 Process control mode**

Elements in process control mode are shown in Table 302.

**Table 302 – Data type: Process control mode**

<b>Standard data type name: Process control mode</b>		
<b>Standard data type code: 69</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
Target	1	Type: Bitstring, representing Process control mode bitstring Size: 1 octet Classification: Static Accessibility: Read/write Default value: OOS Valid value set : See Table 303
Actual	2	Type: Bitstring, representing Process control mode bitstring Size: 1 octet Classification: Dynamic Accessibility: Read only Default value: OOS Valid value set : See Table 303
Permitted	3	Type: Bitstring, representing Process control mode bitstring Size: 1 octet Classification: Static Accessibility: Read/write Default value: OOS Valid value set : See Table 303
Normal	4	Type: Bitstring, representing Process control mode bitstring Size: 1 octet Classification: Static Accessibility: Read/write Default value: OOS Valid value set : See Table 303

The value of each element of the mode structure is represented by a bitstring containing the bits in Table 303, where the <reserved> bits shall be set to zero (0).

**Table 303 – Data type: Process control mode bitstring**

<b>Bit 7 (MSB)</b>	<b>Bit 6</b>	<b>Bit 5</b>	<b>Bit 4</b>	<b>Bit 3</b>	<b>Bit 2</b>	<b>Bit 1</b>	<b>Bit 0 (LSB)</b>
<reserved>	<reserved>	<reserved>	AUTO	MAN	<reserved>	<reserved>	OOS

That is:

- OOS is equal to 0x01, with the equivalent decimal value of 1.
- MAN is equal to 0x08, with the equivalent decimal value of 8.
- AUTO is equal to 0x10, with the equivalent decimal value of 16.

#### 12.20.6 Scaling

Elements in process control scaling are shown in Table 304.

**Table 304 – Data type: Process control scaling**

<b>Standard data type name: Process control scaling</b>		
<b>Standard data type code: 68</b>		
Element name	Element identifier	Element scalar type
Engineering units at 100% (the upper range of the associated object parameter)	1	Type: IEEE 754_Float32 Size: 4 octets Classification: Static Accessibility: Read/write Default value : 0 Valid value set: See IEEE 754
Engineering units at 0% (the lower range of the associated object parameter)	2	Type: IEEE 754_Float32 Size: 4 octets Classification: Static Accessibility: Read/write Default value : 0 Valid value set: See IEEE 754
Units index as defined in Foundation Fieldbus.  NOTE See Foundation Fieldbus Technical Device 16 (document TN-016, rev 3.0 dated Nov 30, 2005) for definition of the unit indices used by the standard.	3	Type: Unsigned16 Size: 2 octets Classification: Static Accessibility: Read/write Default value : Not specified Valid value set: 0 to 65535
Decimal point location (represents the number of places to the right of the decimal point, i.e., the significance of fractional part of the associated value)	4	Type: Unsigned8 Size: 1 octet Classification: Static Accessibility: Read/write Default value : 0 Valid value set: 0 to 255

## 12.21 Additional tables

### 12.21.1 Process control profile standard objects

Table 305 lists process control profile standard objects.

**Table 305 – Process control standard objects**

Object type	Defined by	Standard object type identifier
Analog input	ISA	99
Analog output	ISA	98
Binary input	ISA	97
Binary output	ISA	96

### 12.21.2 Services

Table 306 provides a list of services.

**Table 306 – Services**

Service	Use
Read	Read the value of one or more attributes from one or more objects of a UAP
Write	Write the value to one or more attributes of one or more objects of a UAP
Execute	Execute a set of functions on object instances of a UAP
Publish	Periodically publish data to subscribers
AlertReport	Report one or more alert conditions
AlertAcknowledge	Acknowledge an AlertReport
Tunnel	Pass payload through

## 12.22 Coding

### 12.22.1 General

The conditions for encoding wireless APDUs in this standard include the following considerations:

- Some messages occur often, such as periodic publications.
- Messages should be short, to preserve battery power.
- There should be minimal selection of ASL service types.

The maximum size of an APDU (which is a TSDU) is determined by subtracting (the size of the information TL adds to the TSDU to form the TPDU) from (the Assigned\_Max\_NSDU\_Size), where Assigned\_Max\_NSDU\_Size is an output parameter of the method used to establish a communication contract. That is:

$$\text{maxAPDUSize} = \text{Assigned\_Max\_NSDU\_Size} - \text{sizeOF(TLInfoAddedtoAPDU)}$$

See 6.3.11.2.5 for further details of communication contract establishment.

ASL coding shall ensure that the maximum APDU size is not exceeded.

**NOTE** Implementers may wish to consult IETF RFC 2348 regarding recommendations for a maximum size communications.

The coding rules defined below represent bit 0 as the least significant bit (LSB) in the value represented.

### 12.22.2 Coding rules for application protocol data units

#### 12.22.2.1 General

All APDUs contain an AL header, and a service type-specific APDU content. Table 307 indicates the general coding construct of an APDU.

**Table 307 – Application messaging format**

May be 1, 2, 3, or 5 octets (see 12.22.2.3)	N octets
ASDU header (ensures routing to correct objects; provides service type identification)	Service-specific content

#### 12.22.2.2 Concatenation

APDUs can be concatenated together, and the concatenation of individual APDUs may be given to the transport layer as a single TSDU. The length of this TSDU shall not exceed the maximum APDU size for communications relative to the corresponding communication contract between the sending and receiving applications.

Table 308 describes the format of concatenated APDUs in a single TSDU.

**Table 308 – Concatenated APDUs in a single TSDU**

APDU_1	...	APDU_n
--------	-----	--------

**NOTE** How the ASL determines when and what to concatenate is a local matter.

#### 12.22.2.3 Application layer header

The AL header supports four object identifier addressing modes which determine header construction beyond the first octet. The object identifier addressing modes are:

- Four-bit object identifier addressing mode;
- Eight-bit (1 octet) object identifier addressing mode;
- Sixteen-bit (2 octets) object identifier addressing mode; and

- Inferred addressing mode, which may be used only in a TSDU that contains concatenated APDUs.

Identification of the UAP containing the object is a function of the transport layer.

The object identifier addressing mode in use for APDU interpretation is indicated in bits 5 and 6 bits of the first octet of the APDU header. Table 309 represents the coding of this first APDU header octet.

**Table 309 – Object addressing**

octets	bits								
		6	5	4	3	2	1	0	
1	Service primitive type (.req = 0 .resp = 1)	7	Object identifier addressing mode 00: 4-bit mode 01: 8-bit mode 10: 16-bit mode 11: inferred mode		ASL service type				

#### 12.22.2.4 Object identifier coding

##### 12.22.2.4.1 General

In all header constructions, the source object identifier represents the object identifier in the application that is initiating the ASL service primitive (that is, the initiator of a .request or a .response primitive), and the destination object identifier represents the object identifier in the application that is receiving the ASL service primitive (that is, the recipient of an .indication or a .confirmation primitive).

##### 12.22.2.4.2 Four-bit object identifier addressing mode

Four-bit object identifier addressing mode indicates that the source object identifier and the destination object identifier each can be expressed in a 4-bit unsigned integer. This addressing mode provides for optimal header compaction for application processes with a small number of objects. This mode is described in Table 310.

**Table 310 – Four-bit addressing mode APDU header construction**

octets	bits								
		7	6	5	4	3	2	1	0
1	Service primitive type	0	0	ASL service type					
2	Source object identifier							Destination object identifier	

##### 12.22.2.4.3 Eight-bit object identifier addressing mode

Eight-bit object identifier addressing mode indicates that the source object identifier and the destination object identifier each can be expressed in an 8-bit unsigned integer, and further that at least one of the object identifiers cannot be expressed in a 4-bit unsigned integer. This mode is described in Table 311.

**Table 311 – Eight-bit addressing mode APDU header construction**

octets	bits								
		7	6	5	4	3	2	1	0
1	Service primitive type	0	1	ASL service type					
2	Source object identifier								
3	Destination object identifier								

##### 12.22.2.4.4 Sixteen-bit object identifier addressing mode

Sixteen-bit object identifier addressing mode indicates that the source object identifier and the destination object identifier each can be expressed in a 16-bit unsigned integer, and further that at least one of the object identifiers cannot be expressed in an 8-bit unsigned integer. This mode is described in Table 312.

**Table 312 – Sixteen-bit addressing mode APDU header construction**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Service primitive type	1	0						ASL service type
2	Source object identifier								
3									
4	Destination object identifier								
5									

#### 12.22.2.4.5 Inferred object identifier addressing mode for optimized concatenations

Inferred object identifier addressing mode is an optimization technique used only within a concatenated APDU. The intent of this technique is to save octets transmitted by eliminating redundant source and object identifiers, which can be determined from the most recently parsed APDU contained within the same APDU concatenation.

Inferred object addressing shall not be indicated in the first APDU of a concatenation.

NOTE Any APDU indicating inferred object addressing mode in the first APDU met in ASL parsing is considered a malformed APDU.

An example is included in Table 313.

**Table 313 – Inferred addressing use case example**

APDU_1	APDU_2	APDU_3	APDU_4	APDU_5
00 object identifier addressing mode	11 object identifier addressing mode (indicates use source and destination OIDs from APDU_1)	11 object identifier addressing mode (indicates use source and destination OIDs from APDU_2, which is used the source and destination OIDs from APDU_1)	01 object identifier addressing mode	11 object identifier addressing mode (indicates use source and destination OIDs from APDU_4)
APDU_1 includes: 00 addressing mode; service type; 4-bit source object identifier; 4-bit destination object identifier; service-specific payload	APDU_2 includes: 11 addressing mode; service type; service-specific payload	APDU_3 includes: 11 addressing mode; service type; service-specific payload	APDU_4 includes: 01 addressing mode; service type; 8-bit source object identifier; 8-bit destination object identifier; service-specific payload	APDU_5 includes: 11 addressing mode; service type; service-specific payload

Table 314 describes the construction of the inferred addressing mode APDU header.

**Table 314 – Inferred addressing mode APDU header construction**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Service primitive type	1	1						ASL service type

#### 12.22.2.5 Object attribute coding

##### 12.22.2.5.1 General

Object attribute coding is determined by an attribute identifier format. The format may indicate:

- Six-bit, not indexed: The attribute fits into an Unsigned6 integer, and is not indexed.
- Six-bit, singly indexed: The attribute fits into an Unsigned6 integer, and requires one index. The attribute index is extensible, as indicated by the first bit of the index. If the first

bit of the index is 0, the index is 7 bits in length. If the first bit of the index is 1, the index is 15 bits in length.

- Six-bit, doubly indexed: The attribute fits into an Unsigned6, and requires two indices. The attribute indices are individually extensible; that is, the first index may be 7 or 15 bits in length, and the second index also may be either 7 or 15 bits in length. The length of the index is determined by the first bit of the index. If the first bit of the index is 0, the index is 7 bits in length. If the first bit of the index is 1, the index is 15 bits in length.
- Twelve-bit, not indexed: The attribute fits does not fit into an Unsigned6, but fits into an Unsigned12. The attribute is not indexed.
- Twelve-bit, singly indexed: The attribute fits into an Unsigned12 integer, and requires one index. The attribute index is extensible, as indicated by the first bit of the index. If the first bit of the index is 0, the index is 7 bits in length. If the first bit of the index is 1, the index is 15 bits in length.
- Twelve-bit, doubly indexed: The attribute fits into an Unsigned6, and requires two indices. The attribute indices are individually extensible; that is, the first index may be 7 or 15 bits in length, and the second index also may be either 7 or 15 bits in length. The length of the index is determined by the first bit of the index. If the first bit of the index is 0, the index is 7 bits in length. If the first bit of the index is 1, the index is 15 bits in length.

NOTE Refer to 12.23.1.3 for the definitions of Unsigned6 and Unsigned12.

#### **12.22.2.5.2 Six-bit attribute identifier, not indexed**

Table 315 indicates the coding for a six-bit attribute identifier that is not an indexed or structured attribute.

**Table 315 – Six-bit attribute identifier, not indexed**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Attribute short form value = binary 00	Attribute identifier							

#### **12.22.2.5.3 Six-bit attribute identifier, singly indexed forms**

Table 316 and Table 317 indicate the coding for a six-bit attribute identifier that may be accessed using a single index.

**Table 316 – Six-bit attribute identifier, singly indexed, with seven-bit index**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Attribute short form value = binary 01	Attribute identifier							
2	0	Index							

**Table 317 – Six-bit attribute identifier, singly indexed, with fifteen-bit index**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Attribute short form value = binary 01	Attribute identifier							
2	1	Index (high order 7 bits)							
3		Index (low order 8 bits)							

#### **12.22.2.5.4 Six-bit attribute identifier, doubly indexed forms**

Table 318, Table 319, Table 320, and Table 321 indicate the coding for a six-bit attribute identifier that may be accessed using two indices.

**Table 318 – Six-bit attribute identifier, doubly indexed, with two seven-bit indices**

octets	bits									
	7	6	5	4	3	2	1	0		
1	Attribute short form value = binary 10		Attribute identifier							
2	0	Index 1								
3	0	Index 2								

**Table 319 – Six-bit attribute identifier, doubly indexed, with two fifteen-bit indices**

octets	bits									
	7	6	5	4	3	2	1	0		
1	Attribute short form value = binary 10		Attribute identifier							
2	1	Index 1 (high order 7 bits)								
3	Index 1 (low order 8 bits)									
4	1	Index 2 (high order 7 bits)								
5	Index 2 (low order 8 bits)									

**Table 320 – Six-bit attribute identifier, doubly indexed, with first index seven-bits long and second index fifteen bits long**

octets	bits									
	7	6	5	4	3	2	1	0		
1	Attribute short form value = binary 10		Attribute identifier							
2	0	Index 1								
3	1	Index 2 (high order 7 bits)								
4	Index 2 (low order 8 bits)									

**Table 321 – Six-bit attribute bit attribute identifier, doubly indexed, with first index fifteen bits long and second index seven bits long**

octets	bits									
	7	6	5	4	3	2	1	0		
1	Attribute short form value = binary 10		Attribute identifier							
2	1	Index 1 (high order 7 bits)								
3	Index 1 (low order 8 bits)									
4	0	Index 2								

#### 12.22.2.5.5 Twelve-bit attribute identifier, not indexed

Table 322 indicates the coding for a twelve-bit attribute identifier that is not indexed.

**Table 322 – Twelve-bit attribute identifier, not indexed**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Attribute long form value = binary 11		Attribute long form, index form = binary 00					Attribute identifier (high order 4 bits)	
2	Attribute identifier (low order 8 bits)								

#### 12.22.2.5.6 Twelve-bit attribute identifier, singly indexed coding forms

Table 323 and Table 324 indicate the coding for a twelve-bit attribute identifier that is accessed using a single index.

**Table 323 – Twelve-bit attribute identifier, singly indexed with seven-bit index**

octets	bits										
	7	6	5	4	3	2	1	0			
1	Attribute long form value = binary 11		Attribute long form, index form = binary 01			Attribute identifier (high order 4 bits)					
2	Attribute identifier (low order 8 bits)										
3	0	Index									

**Table 324 – Twelve-bit attribute identifier, singly indexed with fifteen bit identifier**

#### **12.22.2.5.7 Twelve-bit attribute identifier, doubly indexed coding forms**

Table 325, Table 326, Table 327, and Table 328 indicate the coding for a twelve-bit attribute identifier that is accessed using two indices.

**Table 325 – Twelve-bit attribute identifier, doubly indexed with two seven bit indices**

octets	bits										
	7	6	5	4	3	2	1	0			
1	Attribute long form value = binary 11		Attribute long form, index form = binary 10			Attribute identifier (high order 4 bits)					
2	Attribute identifier (low order 8 bits)										
3	0	Index 1									
4	0	Index 2									

**Table 326 – Twelve-bit attribute identifier, doubly indexed with two fifteen bit indices**

**Table 327 – Twelve-bit attribute identifier, doubly indexed with first index seven-bits long and second index fifteen-bits long**

**Table 328 – Twelve-bit attribute identifier, doubly indexed with the first index fifteen bits long and the second index seven bits long**

octets	bits											
	7	6	5	4	3	2	1	0				
1	Attribute long form value = binary 11		Attribute long form, index form = binary 10		Attribute identifier (high order 4 bits)							
2	1	Index 1 (high order 7 bits)										
3	Index 2 (low order 8 bits)											
4	0	Index 2										

**12.22.2.5.8 Reserved for future use**

Table 329 identifies an attribute identifier form that is reserved for future use.

**Table 329 – Twelve-bit attribute identifier, reserved form**

octets	bits								
	7	6	5	4	3	2	1	0	
1	Attribute long form value = binary 11		Attribute long form, index form reserved for future use: binary 11		Reserved for future use				

**12.22.2.6 Read**

Table 330 provides coding rules for the service specific portion of a read service request APDU.

Application Request ID is an identifier that enables the client to match a service response with the original service request. A service response shall include a copy of the Request ID from the corresponding service request.

**Table 330 – Coding rules for read service request**

octets	bits							
	7	6	5	4	3	2	1	0
1	Request ID							
...	Attribute identifier (see coding rules for attribute identifier)							

Table 331 provides coding rules for a read service response with 7-bit length field.

**Table 331 – Coding rules for read service response with seven bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Request ID							
2	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							
3	ServiceFeedbackCode							
4	0	Length – Optional, included only if ServiceFeedbackCode indicates success						
5...n	Value – OPTIONAL, present only if ServiceFeedbackCode only if indicates success							

NOTE Refer to 12.23.3 for the definitions of ServiceFeedbackCode for application layer services.

Table 332 provides coding rules for a read service response with 15-bit length field.

**Table 332 – Coding rules for read service response with fifteen-bit length field**

octets	bits														
	7	6	5	4	3	2	1	0							
1	Request ID														
2	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0														
3	ServiceFeedbackCode														
4	1	Length – Optional, high order 7 bits present only if ServiceFeedbackCode indicates success													
5	Length – Optional, low order 8 bits, present only if ServiceFeedbackCode indicates success														
6...n	Value – OPTIONAL, present only if ServiceFeedbackCode indicates success														

### 12.22.2.7 Write

Table 333 and Table 334 provide coding rules for a write service request.

Application Request ID is an identifier that enables the client to match a service response with the original service request. A service response shall include a copy of the Request ID from the corresponding service request.

**Table 333 – Coding rules for write service request with seven bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Request ID							
...	Attribute identifier (see attribute encoding rules)							
...	0	Length						
...n	Value							

**Table 334 – Coding rules for write service request with fifteen-bit length field**

octets	bits														
	7	6	5	4	3	2	1	0							
1	Request ID														
...	Attribute identifier (see attribute encoding rules)														
...	1	Length (upper 7 bits)													
...	Length (lower 8 bits)														
...n	Value														

Table 335 provides coding rules for a write service response.

**Table 335 – Coding rules for write service response**

octets	bits							
	7	6	5	4	3	2	1	0
1	Request ID							
2	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							
3	ServiceFeedbackCode							

### 12.22.2.8 Execute

Table 336 and Table 337 provide coding rules for an execute service request.

Application Request ID is an identifier that enables the client to match a service response with the original service request. A service response shall include a copy of the Request ID from the corresponding service request.

**Table 336 – Coding rules for execute service request with seven-bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Request identifier							
2	Method identifier							
3	0	Length in octets of request parameters						
4 to n	Request Parameters, OPTIONAL (included if size of length of request parameters is greater than 0)							

**Table 337 – Coding rules for execute service request with fifteen-bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Request identifier							
2	Method identifier							
3	1	Length in octets of response parameters (upper 7 bits)						
4	Length (lower 8 bits)							
5 to n	Response Parameters, OPTIONAL (included if Size of Response Parameters is present, and has value greater than 0)							

Table 338 and Table 339 provide coding rules for an execute service response.

**Table 338 – Coding rules for execute service response with 7-bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Request identifier							
2	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							
3	Forward explicit congestion control echo							
4	0	Length in octets of response parameters						
5 to n	Response Parameters, OPTIONAL (included if Size of Response Parameters is present, and > 0)							

**Table 339 – Coding rules for execute service response with 15-bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Request identifier							
2	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							
3	Forward explicit congestion control echo							
4	1	Length in octets of response parameters (upper 7 bits)						
5	Length (lower 8 bits)							
6 to n	Response Parameters, OPTIONAL (included if Size of Response Parameters is present, and > 0)							

#### 12.22.2.9 Tunnel

Table 340 and Table 341 provide coding rules for a tunnel service request.

**Table 340 – Coding rules for tunnel service request with seven-bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	0	7 bit length						
2 to n	Payload							

**Table 341 – Coding rules for tunnel service request with fifteen-bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	1	Length (upper 7 bits)						
2	Length (lower 8 bits)							
3 to n	Payload							

Table 342 and Table 343 provide coding rules for a tunnel service response.

**Table 342 – Coding rules for tunnel service response with seven-bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							Forward explicit congestion control echo
2	0	7 bit length						
3 to n	Payload							

**Table 343 – Coding rules for tunnel service response with fifteen-bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Reserved for future use by this standard. For compliance with this version of this standard, these bits shall be set to 0							Forward explicit congestion control echo
2	1	Length (upper 7 bits)						
3	Length (lower 8 bits)							
4 to n	Payload							

#### 12.22.2.10 AlertReport

Table 344 and Table 345 provide coding rules for an AlertReport service request.

**Table 344 – Coding rules for AlertReport service with seven bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Alert report ID							
2	Detecting object application process identifier (Transport layer port) – high order 8 bits							
3	Detecting object application process identifier (Transport layer port) – low order 8 bits							
4	Detecting object identifier – high order 8 bits							
5	Detecting object identifier – low order 8 bits							
6 to 11	TAINetworkTimeValue							
12	Class	Direction	Category	Alert Priority				
13	Type							
14	0	Length of data						
15	Data							

**Table 345 – Coding rules for AlertReport service with fifteen-bit length field**

octets	bits							
	7	6	5	4	3	2	1	0
1	Alert report ID							
2	Detecting object application process identifier (Transport layer port) – high order 8 bits							
3	Detecting object application process identifier (Transport layer port) – low order 8 bits							
4	Detecting object identifier – high order 8 bits							
5	Detecting object identifier – low order 8 bits							
6 to 11	TAINetworkTimeValue							
12	Class	Direction	Category	Alert Priority				
13	Type							
14	1	Length of data (high order 7 bits)						
15	Length (lower 8 bits)							
16	Data							

### 12.22.2.11 AlertAcknowledge

Table 346 provides coding rules for an AlertAcknowledge service request.

**Table 346 – Coding rules for AlertAcknowledge service**

octets	bits							
	7	6	5	4	3	2	1	0
1	Alert report ID							

### 12.22.2.12 Publish

Table 347 provides coding rules for a native publish service.

When used in conjunction with a concentrator object, “Data” in the payload comprises the entire data communicated, which is a configured sequence of process control variables. The process control variables include both status information and process values. The structure of the data is indicated by the publishing content version. The freshness sequence number is within the scope of a particular concentrator object.

**Table 347 – Coding rules for publish service for a native sequence of values**

octets	bits							
	7	6	5	4	3	2	1	0
1	Publishing content version							
2	Freshness sequence number							
3 to n	Data							

Table 348 provides coding rules for a publish service used to convey either an internally encoded octet string, or non-native data. Use of this service for non-native data enables support for tunneling.

**Table 348 – Coding rules for publish service – non-native (for tunnel support)**

octets	bits							
	7	6	5	4	3	2	1	0
1 to n	Data							

### 12.22.2.13 Concatenation

Table 349 provides coding rules for constructing a single TSDU which contains multiple logical APDUs.

**Table 349 – Coding rules for concatenate service**

octets	bits							
	7	6	5	4	3	2	1	0
1 to n	SEQUENCE OF APDUS							

### 12.22.3 Coding of application data

#### 12.22.3.1 General

Coding of application data is always primitive. In the tables below, octet 1 represents the most significant octet, bit 7 represents the most significant bit (MSB) within an octet, and bit 0 represents the least significant bit (LSB) within an octet.

The semantics of user data is known by:

- Prior agreement (e.g., tunnel payload content);
- Position in the APDU with fixed field size for content; or
- Existing fields in the APDU.

In these situations, no additional decoding information is added to the APDU.

Coding rules for application data are provided in Table 350 and Table 351.

**Table 350 – General coding rule for size-invariant application data**

Data
------

If the position or size may vary, such as for data type OctetString, then size information is explicitly added to the APDU, as shown in Table 351.

**Table 351 – Coding rules for application data of varying size**

Size of data (in octets) , Unsigned8 (N)	Data (Length N)
--	-----------------

Subsequent clauses of this standard define the content of the Data coding field for the standard data types.

#### 12.22.3.2 Boolean

- Data type: Boolean
- Range: True, False
- Size: 1 octet

Representation of True value is any non-zero value, as shown in Table 352.

**Table 352 – Coding rules for Boolean data – true**

octets	bits							
	7	6	5	4	3	2	1	0
1	One or more bits shall be set to a non-zero value.							

Representation of False value in one octet (value = 0 in hexadecimal notation) is illustrated in Table 353.

**Table 353 – Coding rules for Boolean data – false**

octets	bits							
	7	6	5	4	3	2	1	0
1	0	0	0	0	0	0	0	0

### 12.22.3.3 Integer

#### 12.22.3.3.1 Integer coding

Integers are coded as 2s complement numbers. In 2s complement arithmetic, negative numbers are represent by the twos complement of the absolute value. In this system, zero has a single representation.

In 2s complement, positive numbers are represented as simple binary, and negative 2s complement numbers are represent as the binary number that when added to a positive number of the same magnitude equals zero.

The most significant bit (i.e., bit 7 for an Integer8 value, bit 15 for an Integer16) indicates the sign of the integer, and is therefore called the sign bit. If the sign bit is zero, then the number represented is greater than or equal to zero (i.e., zero, or a positive number). If the sign bit is one, then the number represented is less than zero (i.e., a negative number).

NOTE To calculate the 2s complement of an integer, invert the binary equivalent of the number by changing all of the ones to zeroes and all of the zeroes to ones (also called 1s complement), and then add one.

Example: Form 2s complement of the value 17.

0001 0001 (binary 17)

To form 2s complement:

First: NOT (0001 0001) = 1110 1110 (NOT operation results in inverting the bits)

Then add 1: (1110 1110) + (0000 0001) = 1110 1111 (twos complement = -17)

#### 12.22.3.3.2 Integer8

- Data type: Integer8
- Range:  $-128 < i < 127$
- Size: 1 octet

#### 12.22.3.3.3 Integer16

- Data type: Integer16
- Range:  $-32768 < i < 32767$
- Size: 2 octets

#### 12.22.3.3.4 Integer32

- Data type: Integer32
- Range:  $2^{31} < i < 2^{31} - 1$  (i.e.,  $-2147483648 < i < 2147483647$ )

- Size: 4 octets

#### **12.22.3.4 Unsigned**

##### **12.22.3.4.1 Unsigned8**

- Data type: Unsigned8
- Range:  $0 < i < 255$
- Size: 1 octet

Table 354 provides coding rules for Unsigned8 data.

**Table 354 – Coding rules for Unsigned8**

<b>octets</b>	<b>bits</b>								
	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>	
1	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	

##### **12.22.3.4.2 Unsigned16**

- Data type: Unsigned16
- Range:  $0 < i < 65535$
- Size: 2 octets

Table 355 provides coding rules for Unsigned16 data.

**Table 355 – Coding rules for Unsigned16**

<b>octets</b>	<b>bits</b>								
	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>	
1	$2^{15}$	$2^{14}$	$2^{13}$	$2^{12}$	$2^{11}$	$2^{10}$	$2^9$	$2^8$	
2	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	

##### **12.22.3.4.3 Unsigned32**

- Data type: Unsigned32
- Range:  $0 < i < 4,294,967,295$
- Size: 4 octets

Table 356 provides coding rules for Unsigned32 data.

**Table 356 – Coding rules for Unsigned32**

<b>octets</b>	<b>bits</b>								
	<b>7</b>	<b>6</b>	<b>5</b>	<b>4</b>	<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>	
1	$2^{31}$	$2^{30}$	$2^{29}$	$2^{28}$	$2^{27}$	$2^{26}$	$2^{25}$	$2^{24}$	
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	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	

##### **12.22.3.4.4 Unsigned64**

- Data type: Unsigned64
- Size: 8 octets

Table 357 provides coding rules for Unsigned64 data.

**Table 357 – Coding rules for Unsigned64**

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

**12.22.3.4.5 Unsigned128**

- Data type: Unsigned128
- Size: 16 octets

Table 358 provides coding rules for Unsigned128 data.

**Table 358 – Coding rules for Unsigned128**

octets	bits							
	7	6	5	4	3	2	1	0
1	2 <sup>127</sup>	2 <sup>126</sup>	2 <sup>125</sup>	2 <sup>124</sup>	2 <sup>123</sup>	2 <sup>122</sup>	2 <sup>121</sup>	2 <sup>120</sup>
2	2 <sup>119</sup>	2 <sup>118</sup>	2 <sup>117</sup>	2 <sup>116</sup>	2 <sup>115</sup>	2 <sup>114</sup>	2 <sup>113</sup>	2 <sup>112</sup>
3	2 <sup>111</sup>	2 <sup>110</sup>	2 <sup>109</sup>	2 <sup>108</sup>	2 <sup>107</sup>	2 <sup>106</sup>	2 <sup>105</sup>	2 <sup>104</sup>
4	2 <sup>103</sup>	2 <sup>102</sup>	2 <sup>101</sup>	2 <sup>100</sup>	2 <sup>99</sup>	2 <sup>98</sup>	2 <sup>97</sup>	2 <sup>96</sup>
5	2 <sup>95</sup>	2 <sup>94</sup>	2 <sup>93</sup>	2 <sup>92</sup>	2 <sup>91</sup>	2 <sup>90</sup>	2 <sup>89</sup>	2 <sup>88</sup>
6	2 <sup>87</sup>	2 <sup>86</sup>	2 <sup>85</sup>	2 <sup>84</sup>	2 <sup>83</sup>	2 <sup>82</sup>	2 <sup>81</sup>	2 <sup>80</sup>
7	2 <sup>79</sup>	2 <sup>78</sup>	2 <sup>77</sup>	2 <sup>76</sup>	2 <sup>75</sup>	2 <sup>74</sup>	2 <sup>73</sup>	2 <sup>72</sup>
8	2 <sup>71</sup>	2 <sup>70</sup>	2 <sup>69</sup>	2 <sup>68</sup>	2 <sup>67</sup>	2 <sup>66</sup>	2 <sup>65</sup>	2 <sup>64</sup>
9	2 <sup>63</sup>	2 <sup>62</sup>	2 <sup>61</sup>	2 <sup>60</sup>	2 <sup>59</sup>	2 <sup>58</sup>	2 <sup>57</sup>	2 <sup>56</sup>
10	2 <sup>55</sup>	2 <sup>54</sup>	2 <sup>53</sup>	2 <sup>52</sup>	2 <sup>51</sup>	2 <sup>50</sup>	2 <sup>49</sup>	2 <sup>48</sup>
11	2 <sup>47</sup>	2 <sup>46</sup>	2 <sup>45</sup>	2 <sup>44</sup>	2 <sup>43</sup>	2 <sup>42</sup>	2 <sup>41</sup>	2 <sup>40</sup>
12	2 <sup>39</sup>	2 <sup>38</sup>	2 <sup>37</sup>	2 <sup>36</sup>	2 <sup>35</sup>	2 <sup>34</sup>	2 <sup>33</sup>	2 <sup>32</sup>
13	2 <sup>31</sup>	2 <sup>30</sup>	2 <sup>29</sup>	2 <sup>28</sup>	2 <sup>27</sup>	2 <sup>26</sup>	2 <sup>25</sup>	2 <sup>24</sup>
14	2 <sup>23</sup>	2 <sup>22</sup>	2 <sup>21</sup>	2 <sup>20</sup>	2 <sup>19</sup>	2 <sup>18</sup>	2 <sup>17</sup>	2 <sup>16</sup>
15	2 <sup>15</sup>	2 <sup>14</sup>	2 <sup>13</sup>	2 <sup>12</sup>	2 <sup>11</sup>	2 <sup>10</sup>	2 <sup>9</sup>	2 <sup>8</sup>
16	2 <sup>7</sup>	2 <sup>6</sup>	2 <sup>5</sup>	2 <sup>4</sup>	2 <sup>3</sup>	2 <sup>2</sup>	2 <sup>1</sup>	2 <sup>0</sup>

**12.22.3.5 Float**

Float represents a floating point value. IEEE 754 is employed to represent a floating point value. The IEEE single format consists of three fields: a 23-bit fraction, an 8-bit biased exponent, and a 1-bit sign. These fields are contiguous, as represented in Table 359.

NOTE See IEEE 754 for information regarding short real number representation and range. IEEE NaN (not a number) and INF (infinity) forms are supported.

Table 359 provides coding rules for floating point values.

**Table 359 – Coding rules for Float**

octets	bits							
	7	6	5	4	3	2	1	0
	Exponent (E)							
1	Sign	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$
	(E) Fraction (F)							
2	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$	$2^{-5}$	$2^{-6}$	$2^{-7}$
3	$2^{-8}$	$2^{-9}$	$2^{-10}$	$2^{-11}$	$2^{-12}$	$2^{-13}$	$2^{-14}$	$2^{-15}$
4	$2^{-16}$	$2^{-17}$	$2^{-18}$	$2^{-19}$	$2^{-20}$	$2^{-21}$	$2^{-22}$	$2^{-23}$

**12.22.3.6 Double-precision float**

For double-precision floating-point values, the standard uses the encoding defined by the IEEE 754 standard for normalized double-precision floating-point numbers. The standard defines the following three fields:

- S, the sign of the number. Values 0 and 1 representing positive or negative, respectively. Field size is one (1) bit.
- E, the exponent of the number, in base 2. The exponent is biased by 1 023. Field size is eleven (11) bit.
- F, the fractional part of the numbers mantissa, also in base 2. Field size is fifty-two (52) bit.

This permits a double-precision floating point value to be calculated by the following equation:

- $(-1)^S \cdot 2^{E-Bias} \cdot 1.F$

NOTE See IEEE 754 for information regarding short real number representation and range. IEEE specifications should be referenced concerning encoding of signed zero, signed infinity (overflow), de-normalized numbers (underflow), and NaN.

**Table 360 – Coding rules for double-precision float**

octets	bits							
	7	6	5	4	3	2	1	0
	Exponent (E)							
1	Sign	$2^{10}$	$2^9$	$2^8$	$2^7$	$2^6$	$2^5$	$2^4$
	Exponent (E) (continued)							
2	$2^3$	$2^2$	$2^1$	$2^0$	$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$
	Fraction (F) (continued)							
3	$2^{-5}$	$2^{-6}$	$2^{-7}$	$2^{-8}$	$2^{-9}$	$2^{-10}$	$2^{-11}$	$2^{-12}$
4	$2^{-13}$	$2^{-14}$	$2^{-15}$	$2^{-16}$	$2^{-17}$	$2^{-18}$	$2^{-19}$	$2^{-20}$
5	$2^{-21}$	$2^{-22}$	$2^{-23}$	$2^{-24}$	$2^{-25}$	$2^{-26}$	$2^{-27}$	$2^{-28}$
6	$2^{-29}$	$2^{-30}$	$2^{-31}$	$2^{-32}$	$2^{-33}$	$2^{-34}$	$2^{-35}$	$2^{-36}$
7	$2^{-37}$	$2^{-38}$	$2^{-39}$	$2^{-40}$	$2^{-41}$	$2^{-42}$	$2^{-43}$	$2^{-44}$
8	$2^{-45}$	$2^{-46}$	$2^{-47}$	$2^{-48}$	$2^{-49}$	$2^{-50}$	$2^{-51}$	$2^{-52}$

**12.22.3.7 VisibleString**

- Type: VisibleString
- Range: See ISO 646 and ISO 2375: Defining registration number 2 + SPACE
- Coding: See ISO 646

NOTE 1 See ISO International Register of Coded Character Sets for further details.

NOTE 2 Where maximum lengths are indicated, the length described represents the length of the data value only, and does not include the length value.

Table 361 provides coding rules for VisibleString data.

**Table 361 – Coding rules for VisibleString**

octets	bits							
	7	6	5	4	3	2	1	0
1	Length in octets (N)							
2	First character in string							
3	Second character in string							
...	....							
N+1	Last character in string							

**12.22.3.8 OctetString**

- Type: OctetString
- Coding: Binary

Table 362 provides coding rules for OctetString data.

**Table 362 – Coding rules for OctetString**

octets	bits							
	7	6	5	4	3	2	1	0
1	Length in octets (N)							
2	First octet in string							
...	...							
N+1	Last octet in string							

**12.22.3.9 Bitstring**

- Type: Bitstring
- Coding: Binary
- Size: Only multiples of 8 bits (i.e., multiples of octets) are legal Bitstring sizes.

Table 363 provides the general coding rules for a Bitstring.

**Table 363 – Coding rules for Bitstring**

octet	bits							
	7	6	5	4	3	2	1	0
1	Length in octets (N)							
2	(8xN-1) <sup>th</sup>	(8xN-2) <sup>th</sup>	(8xN-3) <sup>th</sup>	(8xN-4) <sup>th</sup>	(8xN-5) <sup>th</sup>	(8xN-6) <sup>th</sup>	(8xN-17) <sup>th</sup>	(8xN-8) <sup>th</sup> (bit in string)
3	(8xN-9) <sup>th</sup>	(8xN-10) <sup>th</sup>	(8xN-11) <sup>th</sup>	(8xN-12) <sup>th</sup>	(8xN-13) <sup>th</sup>	(8xN-14) <sup>th</sup>	(8xN-15) <sup>th</sup>	(8xN-16) <sup>th</sup>
...								
N+1	etc.							

Table 364 illustrates an example of the coding for a Bitstring of size 8.

**Table 364 – Example of coding for Bitstring of size 8**

octets	bits							
	7	6	5	4	3	2	1	0
1	Length in octets (1)							
2	7	6	5	4	3	2	1	0

#### 12.22.4 Time-related data types

##### 12.22.4.1 TAITimeDifference

In order to keep all time formats similar in this standard, TAITimeDifference is indicated in TAI time format. However, the interpretation of TAITimeDifference is considered a time difference, and is not relative to any TAI start of time instant.

- Data type: TAITimeDifference
- Valid range:
- 0...4 294 967 295 seconds; and
- 0...65 535 TAI second fractional part, in units of  $2^{-16}$  seconds.

Coding consists of an Unsigned32 to represent time in seconds, and an Unsigned16 to represent the fractional part of TAI time. This is represented as a string of 6.

Table 365 provides coding rules for TAITimeDifference.

**Table 365 – Coding rules for TAITimeDifference**

octet	bits								range
	7	6	5	4	3	2	1	0	
1	$2^{31}$	$2^{30}$	$2^{29}$	$2^{28}$	$2^{27}$	$2^{26}$	$2^{25}$	$2^{24}$	TAI time in seconds
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	$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$	$2^{-5}$	$2^{-6}$	$2^{-7}$	$2^{-8}$	Fractional TAI time in units of $2^{-16}$
6	$2^{-9}$	$2^{-10}$	$2^{-11}$	$2^{-12}$	$2^{-13}$	$2^{-14}$	$2^{-15}$	$2^{-16}$	

##### 12.22.4.2 TAINetworkTimeValue

TAINetworkTimeValue represents the network time in TAI time. Four octets represent the current TAI time in seconds, and two octets represent the fractional TAI time in units of  $2^{-16}$  seconds.

- Data type: TAINetworkTimeValue
- Valid range:
- 0 to 4 294 967 295 seconds; and
- 0 to 65 535 TAI second fractional part, in units of  $2^{-16}$  seconds.

Table 366 provides coding rules for TAINetworkTimeValue

**Table 366 – Coding rules for TAINetworkTimeValue**

octet	bits								range
	7	6	5	4	3	2	1	0	
1	$2^{31}$	$2^{30}$	$2^{29}$	$2^{28}$	$2^{27}$	$2^{26}$	$2^{25}$	$2^{24}$	TAI time in seconds
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	$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$	$2^{-5}$	$2^{-6}$	$2^{-7}$	$2^{-8}$	Fractional TAI time in units of $2^{-16}$
6	$2^{-9}$	$2^{-10}$	$2^{-11}$	$2^{-12}$	$2^{-13}$	$2^{-14}$	$2^{-15}$	$2^{-16}$	

##### 12.22.4.3 TAITimeRounded

TAITimeRounded represents the TAI time in integral seconds. Four octets represent the rounded TAI time in seconds.

- Data type: TAITimeRounded

- Valid Range: 0 to 4 294 967 295 seconds

Table 367 provides coding rules for TAITimeRounded.

**Table 367 – Coding rules for TAITimeRounded**

octet	bits								range
	7	6	5	4	3	2	1	0	
1	$2^{31}$	$2^{30}$	$2^{29}$	$2^{28}$	$2^{27}$	$2^{26}$	$2^{25}$	$2^{24}$	TAI time in seconds
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	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$	

#### 12.22.4.4 Standard data structures

Standard data structures are coded by concatenating the coded values for the structure elements in order from the lowest numbered element to the highest numbered element, beginning at octet 1 of the coded result.

#### 12.22.4.5 Null

The data type null has size of zero (0) octets. Typically null is used for semantic consistency representing the potential for content, but with no content identified.

#### 12.22.4.6 Packed

The data type packed indicates that one or more elements of the standard data types have been concatenated together without gap. The composition of packed data is implicitly known by the communicating correspondents.

#### 12.22.4.7 Structured data

##### 12.22.4.7.1 SEQUENCE

SEQUENCE is used to indicate structured data of the same or different standard data type(s). This is akin to a record construct.

Sequences for this standard do not support optional members. If options are identified, a separate sequence (structure) shall be created. The structure of the sequence is required to be implicitly known by the correspondents, and hence is not explicitly conveyed.

##### 12.22.4.7.2 SEQUENCE OF

For data, SEQUENCE OF is used to indicate an array construct. Array content may either be conveyed in entirely, or a specified individual element of an array may be conveyed.

For conveyance of an individual element, the data type of the element is implicitly known by the correspondents. Since some data types are variable in size, the size of the element is conveyed with the element data.

When arrays are conveyed in their entirety, they are encoded in row-major-order. The size of the array in octets shall also be included. The data type of the elements is also known implicitly by the corresponding endpoints, and is not explicitly indicated in the APDU. The dimension(s) of the array is also implicitly known by the corresponding endpoints, and hence is not explicitly included in the APDU.

NOTE Following standard matrix notation, rows are identified by the first index of a two-dimensional array and columns by the second index. For example, an array in the "C" programming language, a two dimensional array defined as

```
int A[2][3] = {{1, 2, 3}, {4, 5, 6}}
```

Encoding would convey in the following order: 1,2,3,4,5,6.

##### 12.22.4.7.3 CHOICE

CHOICE represents a selection chosen from a predefined enumeration of acceptable possibilities. Content of the data varies based on the choice selected.

Choice constructs are neither required nor supported for standard data structures. Separate data structures should be employed, rather than choices within data structures.

## 12.23 Syntax

### 12.23.1 Application protocol data unit

#### 12.23.1.1 Start of containing module

```
( ISA100.11a:2009 (71) ) DEFINITIONS
  IMPLICIT TAGS
  EXPORTS ISA100_TSDU;
  ::= BEGIN
```

#### 12.23.1.2 Top level definitions

```
ISA100_TSDU ::= IMPLICIT CHOICE (
  individualAPDU          ASLIndividualAPDU,
  concatenatedAPDU         ASLConcatenatedAPDU
)
ASLIndividualAPDU ::= IMPLICIT CHOICE(
  confirmedRequestAPDU    ASLConfirmedServiceRequest,
  confirmedResponseAPDU   ASLConfirmedServiceResponse,
  unconfirmedRequestAPDU  ASLUnconfirmedServiceRequest,
  publicationAPDU          ASLPublicationRequest
)
ASLConcatenatedAPDU ::= IMPLICIT SEQUENCE (
  IMPLICIT CHOICE (
    -- implicit based on the content of the APDU header, which is common across the choices
    confirmedRequest          ASLConfirmedServiceRequest,
    confirmedResponse         ASLConfirmedServiceResponse,
    unconfirmedRequest        ASLUnconfirmedServiceRequest
  )
)
```

NOTE This concatenation works because the size of each aperiodic APDU is either determined by explicit information or is implicit by service primitive definition.

#### 12.23.1.3 Common substitutions

Float32 ::= REAL (WITH COMPONENTS(, base(10)) SIZE 32)	-- single-precision
Float64 ::= REAL (WITH COMPONENTS(, base(10)) SIZE 64)	-- double-precision
Integer8 ::= INTEGER (-128...127)	-- 8 bit integer
Integer16 ::= INTEGER(-32768...32767)	-- 16 bit integer
Integer32 ::= INTEGER(-4294967296... 4294967295)	-- 32 bit integer
Unsigned1 ::= INTEGER(0...1)	-- 1 bit unsigned
Unsigned2 ::= INTEGER(0...3)	-- 2 bit unsigned
Unsigned3 ::= INTEGER(0...7)	-- 3 bit unsigned
Unsigned4 ::= INTEGER(0...15)	-- 4 bit unsigned
Unsigned5 ::= Integer(0...31)	-- 5 bi unsigned
Unsigned6 ::= INTEGER(0...63)	-- 6 bit unsigned
Unsigned7 ::= INTEGER(0...127)	-- 7 bit unsigned
Unsigned8 ::= INTEGER(0...255)	-- 8 bit unsigned
Unsigned9 ::= INTEGER(0...511)	-- 9 bit unsigned
Unsigned10 ::= INTEGER(0...1023)	-- 10 bit unsigned
Unsigned11 ::= INTEGER(0...2047)	-- 11 bit unsigned
Unsigned12 ::= INTEGER(0...4095)	-- 12 bit unsigned
Unsigned13 ::= INTEGER(0...8191)	-- 13 bit unsigned
Unsigned14 ::= INTEGER(0...16383)	-- 14 bit unsigned
Unsigned15 ::= INTEGER(0...32767)	-- 15 bit unsigned
Unsigned16 ::= INTEGER(0...65535)	-- 16 bit unsigned
Unsigned32 ::= INTEGER(0...4294967295)	-- 32 bit unsigned
Unsigned63 ::= INTEGER(0...9223372036854775807)	-- 63 bit unsigned
Unsigned64 ::= INTEGER(0...18446744073709551615)	-- 64 bit unsigned
Unsigned128 ::= INTEGER(0...340282366920938463374607431768212455)	-- 128 bit unsigned

```

Uint1 ::= Unsigned1
Uint2 ::= Unsigned2
Uint3 ::= Unsigned3
Uint4 ::= Unsigned4
Uint5 ::= Unsigned5
Uint6 ::= Unsigned6
Uint7 ::= Unsigned7
Uint8 ::= Unsigned8
Uint9 ::= Unsigned9
Uint10 ::= Unsigned10
Uint11 ::= Unsigned11
Uint12 ::= Unsigned12
Uint13 ::= Unsigned13
Uint14 ::= Unsigned14
Uint15 ::= Unsigned15
Uint16 ::= Unsigned16
UInt32 ::= Unsigned32
Unit64 ::= Unsigned64
Uint128 ::= Unsigned128

```

```
Octet1 ::= Unsigned8
```

```

TAITimeDifference ::= SEQUENCE (
    Seconds                      -- not referenced to TAI instant
    Unsigned32,
    FractionOfSeconds            Unsigned16
)
TAINetworkTimeValue ::= SEQUENCE (
    Seconds                      -- referenced to TAI start time instant
    Unsigned32,
    FractionOfSeconds            Unsigned16
)

```

#### **12.23.1.4 Application sub-layer header**

```

RequestResponse ::= Unsigned1 (
    request      (0),
    response     (1)
)

```

```

ObjectAddressingMode ::= Unsigned2 (
    compact       (0)          -- indicates 4 bit object identifiers
    midSize       (1)          -- indicates 8 bit object identifiers
    fullSize      (2)          -- indicates 16 bit object identifiers
    inferred      (3)          -- shall only be used in concatenation, see standard text for use
)

```

```

ASLService ::= Unsigned5 (
    Publish        0,
    AlertReport   1,
    AlertAcknowledge 2,
    Read           3,
    Write          4,
    Execute        5,
    Tunnel         6
)
-- values 7 through 31 are reserved for future use by this standard
)

```

ASLConfirmedServiceRequest ::= CHOICE(

- the first octet of the ConfirmedServiceRequest is constructed with
  - bit 7 (MSB) containing RequestResponse
  - bits 6 and 5 containing ObjectAddressingMode
  - bits 4 through 0 containing ASLService

readCompact	[3]	IMPLICIT ReadRequestPDU,	-- bit pattern: 0 00 00011
readMidSize	[35]	IMPLICIT ReadRequestPDU,	-- bit pattern: 0 01 00011
readFull	[67]	IMPLICIT ReadRequestPDU,	-- bit pattern: 0 10 00011
readInferred	[99]	IMPLICIT ReadRequestPDU,	-- bit pattern: 0 11 00011
writeCompact	[4]	IMPLICIT WriteRequestPDU,	-- bit pattern: 0 00 00100
writeMidSize	[36]	IMPLICIT WriteRequestPDU,	-- bit pattern: 0 01 00100
writeFull	[68]	IMPLICIT WriteRequestPDU,	-- bit pattern: 0 10 00100
writeInferred	[100]	IMPLICIT WriteRequestPDU,	-- bit pattern: 0 11 00100
executeCompact	[5]	IMPLICIT ExecuteRequestPDU,	-- bit pattern: 0 00 00101
executeMidSize	[37]	IMPLICIT ExecuteRequestPDU,	-- bit pattern: 0 01 00101
executeFull	[69]	IMPLICIT ExecuteRequestPDU,	-- bit pattern: 0 10 00101
executeInferred	[101]	IMPLICIT ExecuteRequestPDU,	-- bit pattern: 0 11 00101
tunnelCompact	[6]	IMPLICIT TunnelRequestPDU,	-- bit pattern: 0 00 00110
tunnelMidSize	[38]	IMPLICIT TunnelRequestPDU,	-- bit pattern: 0 01 00110
funnelFull	[70]	IMPLICIT TunnelRequestPDU,	-- bit pattern: 0 10 00110
tunnelInferred	[102]	IMPLICIT TunnelRequestPDU	-- bit pattern: 0 11 00110

)

ASLConfirmedServiceResponse ::= CHOICE(

- the first octet of the ConfirmedServiceResponse is constructed with
  - bit 7 (MSBO containing RequestResponse) = 1 -- only response form is valid
  - bits 6 and 5 containing ObjectAddressingMode
  - bits 4 through 0 containing ASLService

readCompact	[131]	IMPLICIT ReadResponsePDU,	--bit pattern: 10000101
readMidSize	[163]	IMPLICIT ReadResponsePDU,	--bit pattern: 10100011
readFull	[195]	IMPLICIT ReadResponsePDU,	--bit pattern: 11000011
readInferred	[227]	IMPLICIT ReadResponsePDU,	--bit pattern: 11100011
writeCompact	[132]	IMPLICIT WriteResponsePDU,	--bit pattern: 10000100
writeMidSize	[164]	IMPLICIT WriteResponsePDU,	--bit pattern: 10100100
writeFull	[196]	IMPLICIT WriteResponsePDU,	--bit pattern: 11000100
writeInferred	[228]	IMPLICIT WriteResponsePDU,	--bit pattern: 11100100
executeCompact	[133]	IMPLICIT ExecuteResponsePDU,	--bit pattern: 10000101
executeMidSize	[165]	IMPLICIT ExecuteResponsePDU,	--bit pattern: 10100101
executeFull	[197]	IMPLICIT ExecuteResponsePDU,	--bit pattern: 11000101
executeInferred	[229]	IMPLICIT ExecuteResponsePDU,	--bit pattern: 11100101
tunnelCompact	[134]	IMPLICIT TunnelResponsePDU,	--bit pattern: 10000110
tunnelMidSize	[166]	IMPLICIT TunnelResponsePDU,	--bit pattern: 10100110
funnelFull	[198]	IMPLICIT TunnelResponsePDU,	--bit pattern: 11000110
tunnelInferred	[230]	IMPLICIT TunnelResponsePDU,	--bit pattern: 11100110

)

ASLUnconfirmedServiceRequest ::= CHOICE (

- the first octet of the UnconfirmedServiceRequest is constructed with
  - bit 7 (MSBO containing RequestResponse) = 0 -- only request form is valid
  - bits 6 and 5 containing ObjectAddressingMode
  - bits 4 through 0 containing ASLService

alertReportCompact	[1]	IMPLICIT AlertReportRequestPDU,	--bit pattern: 00000001
alertReportMidSize	[33]	IMPLICIT AlertReportRequestPDU,	--bit pattern: 00100001
alertReportFull	[65]	IMPLICIT AlertReportRequestPDU,	--bit pattern: 01000001
alertReportInferred	[97]	IMPLICIT AlertReportRequestPDU,	--bit pattern: 01100001
alertAcknowledgeCompact	[2]	IMPLICIT AlertAcknowledgeRequestPDU,	--bit pattern: 00000010
alertAcknowledgeMidSize	[34]	IMPLICIT AlertAcknowledgeRequestPDU,	--bit pattern: 00100010
alertReportFull	[66]	IMPLICIT AlertAcknowledgeRequestPDU,	--bit pattern: 01000010
alertReportInferred	[98]	IMPLICIT AlertAcknowledgeRequestPDU,	--bit pattern: 01100010
tunnelCompact	[6]	IMPLICIT TunnelRequestPDU,	--bit pattern: 00000110
tunnelMidSize	[38]	IMPLICIT TunnelRequestPDU,	--bit pattern: 00100110
funnelFull	[70]	IMPLICIT TunnelRequestPDU,	--bit pattern: 01000110
tunnelInferred	[102]	IMPLICIT TunnelRequestPDU,	--bit pattern: 01100110

)

```

ASLPublicationServiceRequest ::= CHOICE (
  -- the first octet of the PublicationServiceRequest is constructed with
  -- bit 7 (MSBO containing RequestResponse) = 0 -- only request form is valid for publication)
  -- bits 6 and 5 containing ObjectAddressingMode
  publishCompact          [0]    IMPLICIT PublishRequestPDU,   bit pattern: 00000000
  publishMidSize          [32]   IMPLICIT PublishRequestPDU,  bit pattern: 00100000
  publishFull             [64]   IMPLICIT PublishRequestPDU, bit pattern: 01000000
  -- inferred addressing is not used as there is no concatenation of publications (see concentrator / dispersion
  objects)
)

```

Individual application protocol data units

```

SourceAndDestinationOIDs ::= IMPLICIT SEQUENCE (OCTET ALIGNED)(
  IMPLICIT CHOICE ( -- as determined by objectAddressingMode in bits 5 and 6 of first octet of APDU
    -- source object represents the initiator of the service primitive (.req or .rsp)
    -- destination object represents the recipient of the service primitive (.ind or .cnf)
    compact IMPLICIT PACKED SEQUENCE (
      compactSourceObject     Unsigned4,
      compactDestinationObject Unsigned4
    )
    midSize IMPLICIT SEQUENCE (
      midSizeSourceOID       Unsigned8,
      midSizeDestinationOID Unsigned8
    )
    fullSize IMPLICIT SEQUENCE (
      fullSizeSourceOID      Unsigned16,
      fullSizeDestinationOID Unsigned16
    )
    inferred NULL
  )
)

```

```

ReadRequestPDU ::= IMPLICIT SEQUENCE
  soidDoid    SourceAndDestinationOIDs,
  readRequest ReadRequest
)

```

```

ReadResponsePDU ::= IMPLICIT SEQUENCE
  soidDoid    SourceAndDestinationOIDs,
  readResponse ReadResponse
)

```

```

WriteRequestPDU ::= IMPLICIT SEQUENCE
  soidDoid    SourceAndDestinationOIDs,
  writeRequest WriteRequest
)

```

```

WriteResponsePDU ::= IMPLICIT SEQUENCE
  soidDoid          SourceAndDestinationOIDs,
  writeResponse     WriteResponse
)

```

```

ExecuteRequestPDU ::= IMPLICIT SEQUENCE
  soidDoid          SourceAndDestinationOIDs,
  executeRequest    ExecuteRequest
)

```

```

ExecuteResponsePDU ::= IMPLICIT SEQUENCE
  soidDoid          SourceAndDestinationOIDs,
  executeResponse   ExecuteResponse
)

```

```

TunnelRequestPDU ::= IMPLICIT SEQUENCE
  soidDoid          SourceAndDestinationOIDs,
  tunnelRequest     TunnelRequest
)

```

```

TunnelResponsePDU ::= IMPLICIT SEQUENCE
    soidDoid                               SourceAndDestinationOIDs,
    tunnelResponse                           TunnelResponse
)

AlertReportRequestPDU ::= IMPLICIT SEQUENCE
    soidDoid                               SourceAndDestinationOIDs,
    alertReportRequest                     AlertReportRequest
)

AlertAcknowledgeRequestPDU ::= IMPLICIT SEQUENCE
    soidDoid                               SourceAndDestinationOIDs,
    alertAcknowledgeRequest               AlertAcknowledgeRequest
)

PublishRequestPDU ::= IMPLICIT SEQUENCE
    soidDoid                               SourceAndDestinationOIDs,
    publishRequest                         PublishRequest
)

```

### 12.23.1.5 Periodic application protocol data units

```

PublishRequest ::= IMPLICIT SEQUENCE (
    IMPLICIT CHOICE ( -- implicitly determined by the corresponding application processes
        Isa100NativeValue           IMPLICIT PublishedValue,          -- single published value
        isa100NativeSequence        IMPLICIT PublishedValueSequence, -- sequence of published values
        nonNative                  IMPLICIT NonNativeSequence   -- publication tunnel
    )
)

PublishedValueSequence ::= IMPLICIT SEQUENCE (
    contentVersion                Unsigned8,                   -- version of configuration of content published
    freshvalueSequenceNumber      Unsigned8,                   -- freshness of this set of data
    publishedValues               SEQUENCE OF ProcessValueAndStatus -- values
)

```

NonNativeSequence ::= IMPLICIT OCTET STRING

```

ProcessValueAndStatus ::= IMPLICIT CHOICE ( -- based on publisher and subscriber application configuration
    analog                      AnalogProcessValueAndStatus,
    boolean                     BooleanProcessValueAndStatus
    -- note: these may be extended by other industries, depending on their need
)

```

```

AnalogProcessValueAndStatus ::= IMPLICIT SEQUENCE (
    valueStatus                 ISA100_PV_Status,
    analogProcessValue          Float32
)

```

```

BooleanProcessValueAndStatus ::= IMPLICIT SEQUENCE (
    valueStatus                 ISA100_PV_Status,
    booleanProcessValue         Boolean
)

```

```

ISA100_PV_Status ::= PACKED SEQUENCE (OCTET ALIGNED) ( -- 1 octet (bit field sizes are: 2 + 1 + 3 + 2)
    quality                    PV_Quality,                  -- 2 bits
    reservedSpareBit           Unsigned1,                  -- 1 bit
    IMPLICIT CHOICE ( -- selected by quality
        [0] BadValueSubstatus   BadValueSubstatus,        -- 3 bits
        [1] UncertainValueSubstatus UncertainValueSubstatus, -- 3 bits (same 3 as above)
        [2] GoodValueSubstatus  GoodValueSubstatus,        -- 3 bits (same 3 as above)
        -- choice [3] is reserved
    ),
    limitStatus                LimitStatus,                -- 2 bits control anti-windup
    information
)

```

```

PV_Quality ::= Unsigned2 ( -- 2 bits
    badValue,                      (0),
    uncertainValue,                 (1),
    goodValue,                     (2)
    -- value (3) is reserved for future use
)
spare                                         -- 1

BadValueSubstatus ::= Unsigned3 ( -- 3 bits
    badValue_NonSpecific,          (0),
    badValue_ConfigurationError,   (1),
    badValue_NotConnected,         (2),
    badValue_DeviceFailure,        (3),
    badValue_SensorFailure,        (4),
    badValue_NoCommunicationWithLUV (5),
    badValue_NoCommunicationNoLUV  (6),
    badValue_OutOfService          (7)
)
                                         -- no spares

UncertainValueSubstatus ::= Unsigned3 ( -- 3 bits
    uncertainValue_NonSpecific,     (0),
    uncertainValue_LastUsableValue,  (1),
    uncertainValue_SubstitutedOrManualEntry (2),
    uncertainValue_InitialValue,     (3),
    uncertainValue_SensorConversionInaccurate, (4),
    uncertainValue_RangeLimitsExceeded (5),
    uncertainValue_SubNormal        (6),
    -- value 7 is reserved for future use
)
                                         -- 1 spare

GoodValueSubstatus ::= Unsigned3 (      -- 3 bits
    goodValue_NoSpecialConditionsExist (0)
    -- value 1 is reserved for future use
    -- value 2 is reserved for future use
    -- value 3 is reserved for future use
    -- value 4 is reserved for future use
    -- value 5 is reserved for future use
    -- value 6 is reserved for future use
    -- value 7 is reserved for future use
)
                                         -- 7 spares

LimitStatus ::= Unsigned2 ( -- 2 bits
    notLimited,                    (0),
    lowLimited,                   (1),
    highLimited,                  (2),
    constant,                     (3)
)
                                         -- both high limited and low limited
                                         -- no spares

highLowLimited LimitStatus ::= LimitStatusconstant      -- alternative symbolic name

lowHighLimited LimitStatus ::= LimitStatusconstant      -- alternative symbolic name

```

### **12.23.1.6 Aperiodic application protocol data units**

CompactObjectIdentifier ::= Unsigned4

MidSizeObjectIdentifier ::= Unsigned8

FullSizeObjectIdentifier ::= Unsigned16

```

ExtensibleInteger ::= IMPLICIT SEQUENCE (OCTET ALIGNED) (
    format                  Boolean, -- 1 bit, set to 0 for short form and to 1 for long form
    IMPLICIT CHOICE ( -- choice is established by the longformat field
        shortForm            Unsigned7,          -- 7 bits      -- value shall be < 0x80
        longForm              Unsigned15,         -- 15 bits     -- value
        shall be ≥ 0x80 and
        )                     -- <0x800; value < 0x80
    invalid
)
)
)

```

An ExtensibleInteger shall use a minimal-size encoding. Use of a longForm to encode a value that could be encoded as a shortForm is invalid and shall be rejected as a protocol error.

```

AttributeOptionType ::= Unsigned2 ( -- code points for attribute option
    sixBitNoIndexing      (0),   -- 6 bit attribute identifier, no index
    sixBitOneDimension    (1),   -- 6 bit attribute identifier, one index (8 or 16 bits)
    sixBitTwoDimensions   (2),   -- 6 bit attribute identifier, two indices (each 8 or 16 bits)
    twelveBitExtended     (3)    -- 12 bit attribute identifier
)

```

```

TwelveBitIndexOption ::= Unsigned2 ( - code points for 12 bit AID indexing option
    twelveBitNoIndexing    (0),
    twelveBitOneDimension  (1),
    twelveBitTwoDimensions (2),
    twelveBitReserved      (3)
)

```

```

ExtensibleAttributeIdentifier ::= IMPLICIT PACKED SEQUENCE (OCTET ALIGNED) (
    attributeFormat        AttributeOptionType --2 bits
    IMPLICIT CHOICE ( -- choice is established by element attributeFormat
        sixBitNoIndexing      Unsigned6,
        sixBitOneDimension IMPLICIT SEQUENCE (OCTET ALIGNED) (
            sixBitOneIndexAID   Unsigned6,
            sixBitOneIndex       ExtensibleInteger,
        )
        sixBitTwoDimensions IMPLICIT SEQUENCE (OCTET ALIGNED) (
            sixBitTwoIndexAID   Unsigned6,
            sixBitTwoIndexNo1    ExtensibleInteger,
            sixBitTwoIndexNo2    ExtensibleInteger
        )
        twelveBitExtended IMPLICIT SEQUENCE (OCTET ALIGNED) (
            twelveBitIndexOption TwelveBitIndexOption,
            twelveBitAID         Unsigned12
            CHOICE ( -- choice is established by the twelveBitIndexOption
                twelveBitNoIndexing NULL,
                twelveBitOneDimension : ExtensibleInteger,
                twelveBitTwoDimensions IMPLICIT SEQUENCE (OCTET ALIGNED) (
                    twelveBitTwoIndexNo1  ExtensibleInteger,
                    twelveBitTwoIndexNo2  ExtensibleInteger
                )
            )
        )
    )
)

```

The four bits in the first octet and eight bits of the second octet of the attributeID are concatenated together form a longer Unsigned12 value if the twelve bit AID option is indicated. The four bits in the first octet are the most significant, and the eight bits in the second octet are the least significant.

```

ScalarType ::= ENUMERATED (
    Null                               (0)
    Boolean                            (1),
    Integer8                           (2),
    Integer16                          (3),
    Integer32                           (4),
    Unsigned8                          (5),
    Unsigned16                         (6),
    Unsigned32                         (7),
    Float32                            (8),
    VisibleString                      (9),
    OctetString                        (10),

    Reserved_11                         (11),
    Reserved_12                         (12),
    Reserved_13                         (13),
    Bitstring                           (14),
    Reserved_15                         (15),
    Reserved_16                         (16),
    Reserved_17                         (17),
    Reserved_18                         (18),
    Reserved_19                         (19),
    Reserved_20                         (20),
    Reserved_21                         (21),
    Reserved_22                         (22),
    Reserved_23                         (23),
    Reserved_24                         (24),
    Reserved_25                         (25),
    Reserved_26                         (26),
    Reserved_27                         (27),
    Reserved_28                         (28),

    Reserved_29                         (29),
    DoublePrecisionFloat                (30),
    TIATimeDifference                  (31),
    TIANetworkTimeValue                (32)
    -- any reserved fields, and any additions are reserved for this standard
)

```

Primitive encoding shall be used for ScalarData, ArrayData, and StructureData value elements. No type information is included in the encoding.

```

GenericSizeAndValue ::= IMPLICIT SEQUENCE OF CHOICE(
    SizeInOctets          ExtensibleInteger,      -- necessary for parsing (e.g., concatenations)
    DataValue              IMPLICIT SEQUENCE OF octet1
)

```

ServiceFeedbackCodeGenericSizeAndValue ::= GenericSizeAndValue

### 12.23.2 Alert reports and acknowledgements

```

AlertClass ::= Unsigned1 ( -- 1 bit
    event                               (0),
    alarm                               (1)
)

```

```

AlertCategory ::= Unsigned2 ( -- 2 bits
    deviceDiagnostic                   (0),
    communicationsDiagnostic           (1),
    security                           (2),
    process                            (3)
)

```

```

AlarmDirection ::= Unsigned1 ( -- 1 bit
    returnToNormalOrNoAlarm           (0), -- for alerts, set this value to 0; for alarm returns set this to zero
    inAlarm                            (1)  -- to report an alarm condition, set this value to 1.
)

```

This standard presently does not define standard alerts for the following industry-independent AL-defined objects:

- UAPMO;
- ARO;
- UDO;
- Concentrator;
- Dispersion;
- Tunnel;
- Interface.

```
ASLMO_Communication_Alerts ::= ENUMERATED (
    malformed_APDU      (0)
-- values 1 through 50 are reserved for future use by this standard
-- values 51 through 100 are reserved for future use by standard profiles
-- vendor-specific codes range 101 through 255
)

AI_ProcessAlerts ::= ENUMERATED ( -- 1 octet;
    outOfServiceAlarm          (0),
    highAlarm                  (1),
    highHighAlarm              (2),
    lowAlarm                   (3),
    lowLowAlarm                (4),
    deviationLowAlarm          (5),
    deviationHighAlarm         (6),

-- values 7 through 50 are reserved for future use by this standard
-- values 51 through 100 are reserved for future use by standard profiles
-- vendor-specific codes range 101 through 255
)

AO_ProcessAlerts ::= ENUMERATED ( -- 1 octet;
    outOfServiceAlarm          (0),
    highAlarm                  (1),
    highHighAlarm              (2),
    lowAlarm                   (3),
    lowLowAlarm                (4),
    deviationLowAlarm          (5),
    deviationHighAlarm         (6),

-- values 7 through 50 are reserved for future use by this standard
-- values 51 through 100 are reserved for future use by standard profiles
-- vendor-specific codes range 101 through 255
)

BI_ProcessAlerts ::= ENUMERATED ( -- 1 octet;
    outOfServiceAlarm          (0),
    discreteAlarm               (1),

-- values 2 through 50 are reserved for future use by this standard
-- values 51 through 100 are reserved for future use by standard profiles
-- vendor-specific codes range 101 through 255
)

BO_ProcessAlerts ::= ENUMERATED (
    outOfServiceAlarm          (0),
    discreteAlarm               (1),

-- values 2 through 50 are reserved for future use by this standard
-- values 51 through 100 are reserved for future use by standard profiles
-- vendor-specific codes range 101 through 255
)

ARMO_Alerts ::= ENUMERATED (
    AlarmRecoveryStart          (0),
    AlarmRecoveryEnd            (1)

-- values2 through 50 are reserved for future use by this standard
-- values 51 through 100 are reserved for future use by standard profiles
-- vendor-specific codes range 101 through 255
)

IndividualAlertID ::= Unsigned8           -- unique ID associated with an individual alert. Assigned by the application process in the UAL.
```

```

statusSignalNamur107 ::= ::= Unsigned8 (
    failure                      (0)   --
    checkFunction                 (1)   --
    offSpec                      (2)   --
    maintenanceRequired          (3)   --
)

IndividualAlert ::= IMPLICIT PACKED SEQUENCE (OCTET ALIGNED)(
    individualAlertID           IndividualAlertID,
    DetectingObjectTransportLayerPort Unsigned16,
    DetectingObject              Unsigned16,
    DetectingObjectType          Unsigned16,
    detectionTime                TIANetworkTimeValue, -- Uint32 for TAI time in seconds and UInt16 for
                                                fractional TAI seconds in units of 2-15
    alertClass                   AlertClass,        -- 1 bit
    alarmDirection               AlarmDirection, -- 0 for event or alarm return; 1 for alarm report
    alertCategory                AlertCategory,     -- 2 bits
    alertPriority                AlertPriority,    -- 4 bits
    alertType                    Unsigned8,        -- object category and type dependent
    alertValueSize               ExtensibleInteger,
    alertValue                   -- present if alertValueSize > 0
    CHOICE ( -- choice is based on AlertCategory
        deviceDiagnostic IMPLICIT SEQUENCE
        (
            statusSignalNamur107 Unsigned8,
            detailedInformation IMPLICIT SEQUENCE OF Octet1 OPTIONAL
        )
        communicationsDiagnostic IMPLICIT SEQUENCE OF Octet1 OPTIONAL
        security IMPLICIT SEQUENCE OF Octet1 OPTIONAL
        process IMPLICIT SEQUENCE OF Octet1 OPTIONAL
    )
    alertValue                  IMPLICIT SEQUENCE OF Octet1 OPTIONAL
                                -- present if alertValueSize > 0
                                -- NAMUR information may be included here for device alerts.
)

```

AlertReportRequest ::= ( -- note: client OID not present; ARMO is implied  
 alert IndividualAlert  
)

AlertAcknowledgeRequest ::= (  
 alertID IndividualAlertID -- server is always ARMO  
)

AlertPriority ::= Unsigned4

Alert priority is a value that indicates the importance of the alert. A larger value implies a more important alert. Host systems map device priorities into host alert priorities that typically include urgent, high, medium, low, and journal. The mapping of alert priority values into these categories are as follows:

- 0 – 2: journal
- 3 – 5: low
- 6 – 8: medium
- 9 – 11: high
- 12 – 15: urgent

MalformedAPDUClass ::= AlertClassevent;

MalformedAPDUAlerCategory ::= AlertCategorycommunicationsDiagnostic

MalformedAPDUAlerType ::= AlertTypemalformedAPDUCommunicationAlert

```

MalformedAPDUAAlertPriority = 7          -- Mid-range of medium priority alerts

MalformedPDUAAlertValueSize ::= 24        -- sizeof(Unsigned128) + sizeof(Unsigned16) +
                                            -- sizeof(TAITimeDifference)
                                            -- where sizeof(TAITimeDifference) = sizeof (Unsigned32) +
                                            -- sizeof(Unsigned16)

MalformedPDUAAlertValue ::= IMPLICIT SEQUENCE ( -- alert value sent by ASL to DMAP
    sourceAddress           Unsigned128,      -- 128 bits to ensure address uniqueness.
    thresholdExceeded       Unsigned16,
    TimeWindow               TAITimeDifference -- Uint32 for TAI time in seconds and Uint16
for
)
                                            fractional TAI seconds in units of 2^-15

```

### 12.23.3 Service feedback code

**NOTE** Service feedback code is used to indicate status (e.g., success), warning (e.g., value limited), or error (e.g., incompatible mode).

ServiceFeedbackCode ::= Unsigned8 (	-- 1octet
-- standard error codes, range 0 – 127	
success	(0) -- success
failure	(1) -- generic failure
other	(2), -- reason other than that listed in this enumeration
invalidArgument	(3), -- invalid attribute to a service call
invalidObjectID	(4), -- invalid object ID
invalidService	(5), -- unsupported or illegal service
invalidAttribute	(6), -- invalid attribute index
invalidElementIndex	(7), -- invalid array or structure element index (or indices)
readOnlyAttribute	(8), -- read-only attribute
valueOutOfRange	(9), -- value is out of permitted range
inappropriateProcessMode	(10), -- process is in an inappropriate mode for the request
incompatibleMode	(11), -- value is not acceptable in current context
invalidValue	(12), -- value (data) not acceptable for other reason (e.g., too large, too small, invalid engineering units code)
internalError	(13), -- device internal problem
invalidSize (14),	-- size is not valid (may be too big or too small)
incompatibleAttribute	(15), -- attribute not supported in this version
invalidMethod	(16), -- invalid method identifier
objectStateConflict	(17), -- state of object in conflict with action requested
inconsistentContent	(18), -- the content of the service requested is inconsistent
invalidParameter	(19), -- value conveyed is not legal for method invocation
objectAccessDenied	(20), -- object is not permitting access
typeMismatch	(21), -- data not as expected (e.g., too many or too few octets)
deviceHardwareCondition	(22), -- device specific hardware condition prevented request from succeeding (e.g., memory defect problem)
deviceSensorCondition	(23), -- problem with sensor detected
deviceSoftwareCondition	(24), -- device specific software condition prevented request from succeeding (e.g., local lockout, local write protection, simulating in progress)
fieldOperationCondition	(25), -- field specific condition prevented request from succeeding (e.g., lockout, or environmental condition not in range)
configurationMismatch	(26), -- a configuration conflict was detected
insufficientDeviceResources	(27), -- e.g., queue full, buffers / memory unavailable
valueLimited	(28), -- e.g., value limited by device
dataWarning	(29), -- e.g., value has been modified due to a device specific reason
invalidFunctionReference	(30), -- function referenced for execution is invalid
functionProcessError	(31), -- function referenced could not be performed due to a device specific reason

warning  alert. writeOnlyAttribute operationAccepted invalidBlockSize invalidDownloadSize unexpectedMethodSequence timingViolation operationIncomplete successful invalidData  dataSequenceError  operationAborted invalidBlockNumber blockDataError blockNotDownloaded downloaded writeProtected invalidMode	(32), -- successful, the there is additional information that may be of interest to the user. Such additional may, for example be conveyed via accessing an attribute, or by sending an alert. (33), -- write-only attribute (e.g., a command attribute) (34), -- method operation accepted (35), -- upload or download block size not valid (36), -- total size for upload not valid (37), -- required method sequencing has not been followed (38), -- object timing requirements have not been satisfied (39), -- method operation, or method operation sequence not (40), -- data received is not valid (e.g., checksum error, data content not as expected, etc.)  (41), -- data is ordered; data received is not in the order required -- example: duplicate data was received (42), -- operation aborted by server (43), -- invalid block number (44), --error in block of data, example, wrong size, invalid content (45), -- the specified block of data has not been successfully (46), -- data is write protected, so write operation is invalid (47), -- operation did not succeed due to invalid mode  -- range 48 through 127 is reserved for future use of this standard  -- vendor-specific device-specific feedback codes, range 128 – 255 vendorDefinedError_128 (128), -- redefinable by each device vendor for each device type ... vendorDefinedError_254 extensionCode feedback code value )
--	--

#### 12.23.4 Read, write, and execute

```

RequestID ::= Unsigned8
ReadRequest ::= IMPLICIT SEQUENCE (
    requestID           RequestID,
    targetAttribute     ExtensibleAttributelIdentifier
)
ApduResponseControlData ::= PACKED IMPLICIT SEQUENCE (
    ForwardCongestionNotificationEcho   Unsigned1,          -- 1 if congestion in forward (request) path was
detected (BECN)                                -- 0 if congestion in the forward (request) path was not detected.
    Spare Unsigned7
)
ReadResponse ::= IMPLICIT SEQUENCE (
    requestID           RequestID,          -- matches corresponding ReadRequest
    apduControl         ApduResponseControlData,
    readValue           ServiceFeedbackCodeGenericSizeAndValue
)
WriteRequest ::= IMPLICIT SEQUENCE (
    requestID           RequestID,
    targetAttribute     ExtensibleAttributelIdentifier
    value               GenericSizeAndValue
)
WriteResponse ::= IMPLICIT SEQUENCE (
    requestID           RequestID,          -- matches corresponding WriteRequest
    apduControl         ApduResponseControlData,
    serviceFeedbackCode ServiceFeedbackCode
)

```

```

MethodInvocationRequest ::= IMPLICIT SEQUENCE (
    methodID                  Unsigned8,
    requestParametersSize     ExtensibleInteger,
    requestParameters          IMPLICIT SEQUENCE of Octet1 OPTIONAL
        -- primitive encoding; data type known by correspondents
        -- requestParameters only present if requestParametersSize >0
)
)

MethodInvocationResponse ::= IMPLICIT SEQUENCE (
    responseParametersSize    ExtensibleInteger,
    responseParameters         IMPLICIT SEQUENCE of Octet1 OPTIONAL
        -- primitive encoding; data type known by correspondents
        -- responseParameters only present if responseParametersSize >0
)
)

ExecuteRequest ::= IMPLICIT SEQUENCE (
    requestID                 RequestID,
    methodInvocationRequest    MethodInvocationRequest -- data type(s) specified by standard
)
)

ExecuteResponse ::= IMPLICIT SEQUENCE (
    requestID                 RequestID,
    apduControl               ApduResponseControlData,
    serviceFeedbackCode        ServiceFeedbackCode,
    methodInvocationResponse   MethodInvocationResponse -- data type(s) specified by standard
)
)

```

### 12.23.5 Tunnel

```

TunnelRequest ::= IMPLICIT SEQUENCE (
    length          ExtensibleInteger,
    tunnelPayload   SEQUENCE OF Octet1
)

```

```

TunnelResponse ::= IMPLICIT SEQUENCE (
    apduControl      ApduResponseControlData,
    length           ExtensibleInteger,
    tunnelPayload   SEQUENCE OF Octet1
)

```

### 12.23.6 End of contained module

END

## 12.24 Detailed coding examples (INFORMATIVE)

### 12.24.1 Read

Scenario: Client object 11 wishes to read data from server object 12, attribute 3. The response indicates the read is successful and returns a value of length two octets.

Table 368 illustrates an example of a request to read multiple values.

**Table 368 – Coding example: Read request for a non-indexed attribute**

Encoding of octets in hexadecimal	Semantic
03	Read request
BC	Client (source)object ID = 11 <sub>10</sub> Server (destination) object ID 12 <sub>10</sub>
XX	Request identifier
03	Attribute ID = 3 (attribute is scalar)

Table 369 illustrates an example of a response to a request to read multiple values.

**Table 369 – Coding example: Read response (corresponding to request contained in the preceding table)**

Encoding of octets in hexadecimal	Semantic
83	Read response
CB	Server (source) object iD = $12_{10}$ Client (destination) object ID = $11_{10}$
XX	Request identifier (Same value as for Request identifier that was included in the corresponding service request).
00	Success
02	Value is two octets long
YY YY	Value

### 12.24.2 Tunnel

Scenario: Object 16 in the client is sending a message to object 20 in the server. The content of the message is to be passed through to the server object.

Table 370 illustrates an example of a tunnel service request that has payload size of 9 octets.

**Table 370 – Coding example: Tunnel service request**

Encoding of octets in hexadecimal	Semantic
06	Tunnel request
09	Length fits into 7 bits
ZZ.....ZZ	Data being tunneled

## 13 Gateway

### 13.1 General

#### 13.1.1 Overview

The primary purpose of a gateway as defined by this standard is to enable host-level applications to interact with wireless field devices. There exists a large installed base of these applications, including automation devices, controllers, and supervisory systems. These applications utilize numerous legacy protocols and require protocol translation in order to interact with the field devices. Such protocol translation may be present in the gateway and also in adapters to legacy wired field devices. Within this standard, the term adapter is used to identify devices that convert from a wired fieldbus protocol to a wireless fieldbus protocol on behalf of one or more field devices.<sup>7</sup> The protocol translation generally serves to tunnel a foreign protocol across a wireless network as described in this standard or to convert a legacy protocol to and from this standard's native format. The term native field device refers to a field device that functions exclusively through the usage of native objects, native services, and native message content as described in this standard.<sup>8</sup> Host-level applications may also be newly written or modified to utilize the native application protocol directly, reducing or eliminating protocol translation within a gateway.

NOTE The protocols defined in this clause are symmetric, in that these protocols may be applied in both gateways and adapters without modification. In practice, the specific foreign protocol features and the usage of a gateway and an adapter relative to host-level applications and field devices will dictate the subset of the protocols that apply to each.

The description of the gateway relates to provision of the following capabilities:

- Interfacing foreign host-level applications through gateways;
- Directly to native field devices.
- Indirectly to legacy wired field devices through legacy adapters.
- Interfacing host-level applications to multiple wireless systems, including a wireless system as described in this standard and one or more foreign wireless systems, through a single common high side interface within a multimode gateway.

This standard provides supporting functionality for the construction of gateways. It does not provide complete details on how to construct any particular gateway. The gateway clause specifies support functionality for foreign protocol translation needs, but does not specify details on how to perform any specific protocol translation or how to interface to any specific plant network.

Legacy protocols were never intended to operate over wireless networks. They do not access information in a manner that conserves energy, and they are not tolerant of delayed access to sleeping devices. The gateway support functionality of this standard is, in large part, intended to enable the construction of gateways that adapt legacy protocols to the requirements of low-energy-consumption wireless devices.

A gateway is a specialized user application process (UAP). Functionality is provided by AL objects and a gateway high side service interface. The objects use the services of the lower protocol suite layers to support the gateway high side interface functions.

---

<sup>7</sup> Usage of the term adapter is not uniform. Technically, an adapter is an interface from a CPU to a communication channel. Technically, a gateway is an interface from one communication channel to another communication channel, where protocol translation is utilized at one or more layers of the protocol suite. There is a precedent set in the automation industry to (incorrectly) use the term adapter to identify devices that convert from a wired fieldbus protocol to a wireless fieldbus protocol on behalf of one or more field devices. There is also a precedent in the automation industry to (correctly) use the term gateway to identify devices that convert from a wireless fieldbus protocol to a wired fieldbus protocol for attachment to a control system. This document adheres to this automation industry usage in an attempt to minimize confusion.

<sup>8</sup> The native tunnel object utilizes the native tunnel and publication services to carry foreign message content. If a field device requires foreign message content to perform its function, then it cannot be considered a native device.

### 13.1.2 Gateway protocol suite diagrams for native devices and adapters

The protocol suite diagram in Figure 17 depicts a gateway interfacing a host-level application (the example control system) to a wireless field device. In this case, the field device is a native device. Protocol translation is performed in the gateway to convert between the plant network protocol and this standard's native protocol. Routers may exist between the gateway and the field device (as depicted in Figure 17), but their operation is transparent.

The protocol suite diagram in Figure 19 depicts a gateway interfacing a host-level application (the example control system) to a wired I/O device through a wireless system as described in this standard. In this case, the I/O device is not a native device, but a legacy device, thus an adapter is required. Protocol translation is performed in both the gateway and in the adapter. The gateway and the adapter convert to and from legacy protocols.

NOTE 1 An adapter to a single legacy wired I/O device could be implemented by performing a protocol translation to and from native formats and without carrying any foreign message content. Such an adapter is indistinguishable from a native I/O device from the viewpoint of a gateway. No special provisions are made for such devices. Additionally, there are no special provisions to facilitate multiplexing for such an adapter where it serves multiple legacy wired I/O devices.

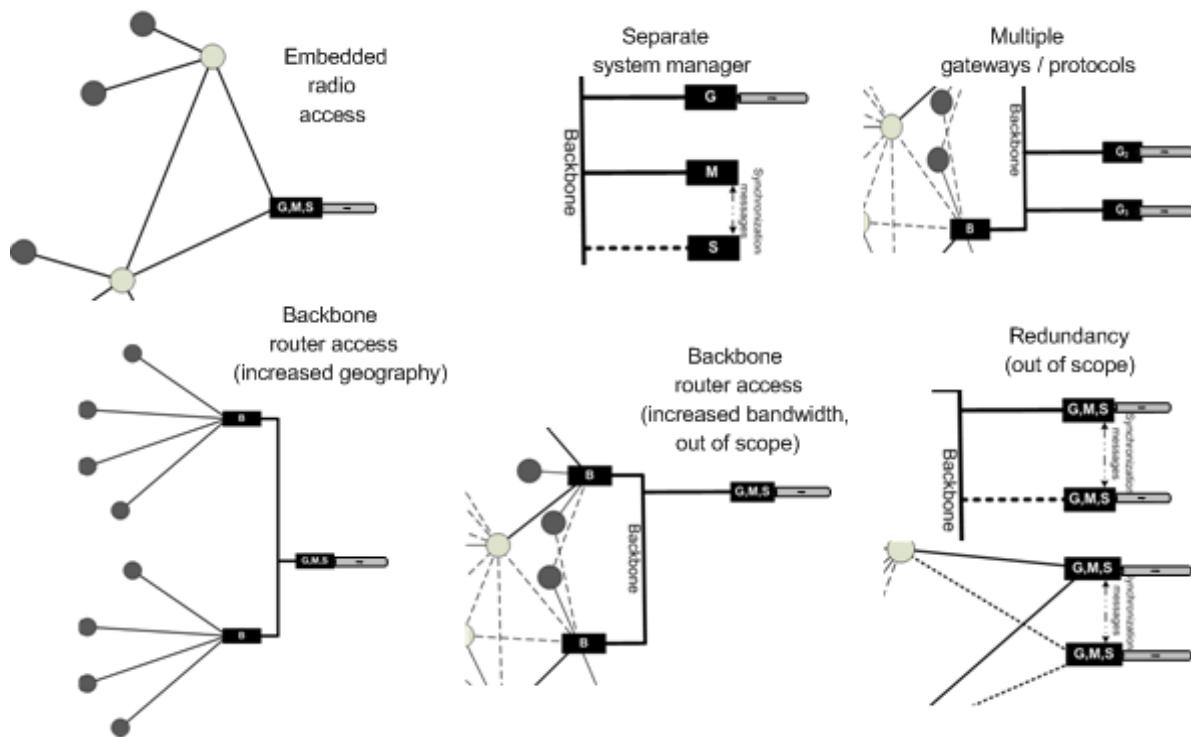
As seen in Figure 19, a gateway and an adapter share a common structure. Both have an interface and protocol suite for a foreign network. Both have an interface and protocol suite as described in this standard. Both have protocol translators. The common structure extends even further - there are common objects and a common high side interface structure. For this reason, no separate role was defined for an adapter. Routers may exist between the gateway and the adapter (as depicted in Figure 19), but their operation is transparent.

NOTE 2 The differences between a gateway and an adapter relate mostly to the implementation. For example, certain legacy protocols only publish from the field and require only producer functionality in the adapter. Others also publish to the field and require produced functionality in both ends. In another example, legacy engineering tools carried into the field and plugged into the legacy network behind the adapter may require the same functions as if they are behind the gateway.

A common protocol translation is required of a gateway and an adapter for them to be interoperable. If the adapter converts to and from native format, it behooves the gateway to do the same. If the adapter tunnels a legacy protocol, the gateway is required to tunnel the same protocol.

### 13.1.3 Gateway scenarios (INFORMATIVE)

Common gateway scenarios are depicted in Figure 135. This figure is not meant to provide an exhaustive description of all variations. Rather, it is included for informational purposes and to illustrate the bounds of the current standard.



**Figure 135 – Gateway scenarios**

As defined in Clause 5, a gateway implements a role within the system. A variety of physical implementations are possible.

Some device implementations may have a radio (PhL and DL) embedded into the same packaging as the gateway, providing direct access to a wireless mesh. Also, some device implementations may have the system management and security management roles co-resident in the gateway, providing a low parts-count solution.

The system manager and security manager may also be separated from the gateway. The gateway does not interact directly with the security manager, but indirectly through the system manager. The gateway acts like any other device in that it requires a communication path to the system manager to become part of an operational system.

The gateway may also communicate to the wireless mesh through backbone routers, where the backbone routers contain radios.

The gateway communication to field devices through backbone routers is transparent in operation. Network layer extensions exist in other portions of this standard to support this transparency. It is, however, necessary to configure this routing within the gateway and the backbone routers. The backbone routers may be used to extend the geographical scope of gateway-connected devices. Backbone routers may also be used to increase bandwidth or to add redundant paths between a gateway and a mesh.

Multiple independent gateways may exist within a system. This is facilitated by independent addressing and independent communication relationships between devices. One use for independent gateways is to support multiple independent protocols. No special provisions are made in this standard for inter-dependent operation between gateways, such as for redundancy or load sharing.

#### 13.1.4 Basic gateway model

Gateways follow the general model depicted in Figure 136.

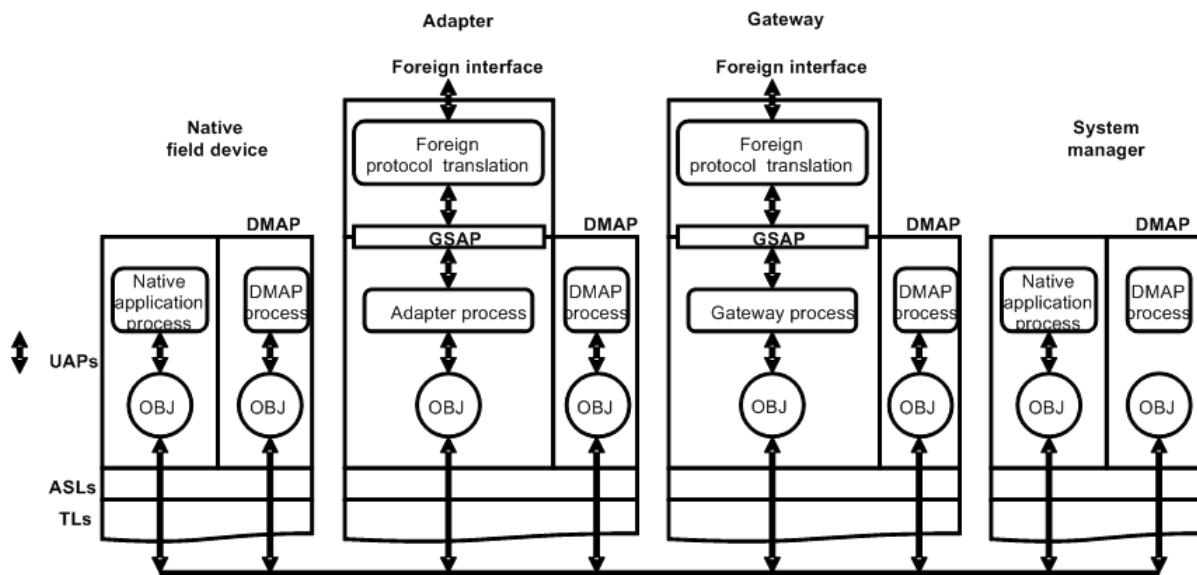


Figure 136 – Basic gateway model

Gateways and adapters host foreign protocol translators that receive and transmit foreign interface messages (typically from a control system, an asset management system, or an engineering system) and use gateway services to interact with wireless devices. Gateway services are provided via a high side interface. This interface is the gateway service access point (GSAP). Protocol translators utilize these services through the GSAP. This standard does not specify protocol translators, only the supporting infrastructure for protocol translators.

Protocol translation conveys application information for control, monitoring, configuration, and management. Foreign protocol messages contain this information. Tunneling, foreign protocol application communication (FPAC), and native object access methods are provided within the objects defined in this standard to support protocol translation as described in Annex N. Each method entails specific tradeoffs of translation effort, energy efficiency, and performance. Practical protocol translators are likely to use a combination of these methods.

Gateways and adapters each have an application process that interfaces to the protocol translators through the GSAP. Each process provides the high side services by utilizing application objects (OBJ). Inter-object communication uses the messaging methods provided through the application sub-layer (ASL).

The GSAP services are used for network management, protocol tunneling, upload and download, alerts, time management, and access to native application and management objects. The GSAP is described in detail in 13.2.

For wired automation devices, an adapter performs a symmetric function and converts foreign interface messages to an adapter high side interface (also a GSAP) via a gateway peer foreign protocol translator.

Gateway access to a native field device (from a gateway) is enabled through the same GSAP; however, the object interactions use native messaging exclusively. A native application process interacts with the gateway or adapter process in a manner that depends on the type of the object.

Gateways, adapters, and native field devices can all be managed through the same symmetric method. The DMAP process is the peer process in this instance. Management is specific to this standard and not to foreign protocols. Foreign protocols may provide additional device and wired fieldbus management methods that are outside the scope of this standard.

The basic gateway model includes the system manager. Access to the system manager by the gateway is necessary in order for the system manager to participate in the system and to perform reporting functions specified in the GSAP.

Backbone routers and routing devices are not shown in Figure 136, because the basic model focuses on the functional messaging interaction between native objects and not on the specifics of routing the messages between objects.

## 13.2 Service access point

### 13.2.1 Summary of services and primitives

The gateway portion of this standard defines a gateway service access point (GSAP) that serves as a high side interface above a wireless communication protocol suite for conveying wireless information and managing wireless behavior. This GSAP is generic and intended for usage as a common interface above the application layer of this standard and above other functionally similar communication protocol suites. Annex P describes implementation of the GSAP services for the wireless protocol suite defined herein by utilization of application layer objects and services. Annex Q describes GSAP service implementation for an alternative wireless protocol suite.

NOTE 1 A primary intent of the GSAP is to allow multimode access where a number of wireless interfaces are available to the gateway. When the devices are geographically located, the GSAP provides consistent reporting information on each underlying wireless to allow improved coexistence. The information includes a report on communication performance to identify potential interference problems, a report on topology to identify collocated devices, and a report on channel and schedule information to identify potential usage conflicts. A series of separate tools, such as the system manager, may provide similar information that is useful for coexistence management.

NOTE 2 Another intent of the GSAP is to provide a standardized model for the configuration and access of multiple underlying wireless networks, reducing the effort for gateway developers where they have multiple fieldbus protocols and wireless systems.

The GSAP interface is usable by a variety of protocol translators to interface to wireless communication protocol suites for conveying wireless information and managing wireless behavior. A protocol translator exists in a gateway. Depending on the implementation, a protocol translator and a GSAP may also exist in an adapter. Protocol translators will vary in complexity depending on the protocol that exists above the gateway and below the adapters. Certain protocols will utilize a subset of the GSAP services. For example, a protocol may only require client/server interaction and not require publish/subscribe services. Functionally, an adapter is considered a subset of a gateway and would only be expected to support a subset of the GSAP interfaces related to conveying wireless information.

NOTE 3 The formats used in this subclause adhere to IEC format conventions. This is in part to allow the IEC to define protocol translators that utilize this standard.

NOTE 4 Since it is outside the scope of this standard to define the protocol translation for specific fieldbus protocols, it is also outside the scope to define the specific subset of GSAP services that each specific fieldbus protocol shall utilize.

The interface services are summarized in Table 371.

**Table 371 – Summary of gateway high side interface services**

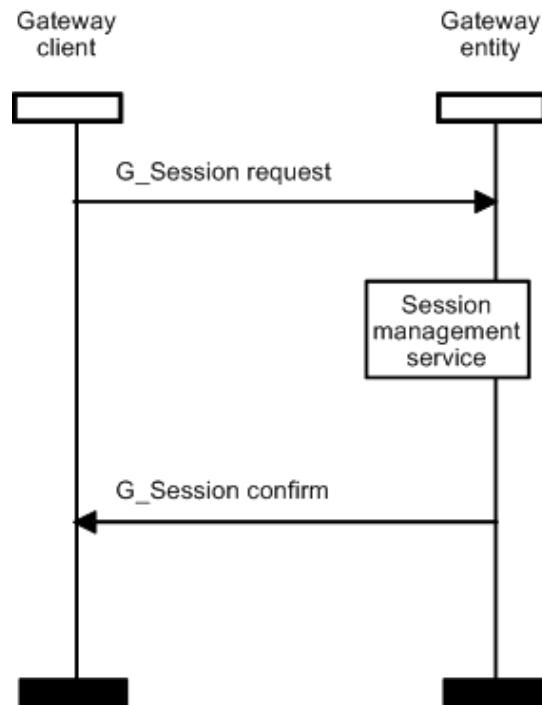
<b>Service</b>	<b>Service subtype</b>	<b>Primitive</b>	<b>Description</b>
Session		G_Session request	
		G_Session confirm	
Lease		G_Lease request	
		G_Lease confirm	
Device_List_Report		G_Device_List_Report request	
		G_Device_List_Report confirm	
Topology_Report		G_Topology_Report request	
		G_Topology_Report confirm	
Schedule_Report		G_Schedule_Report request	
		G_Schedule_Report confirm	
Device_Health_Report		G_Device_Health_Report request	
		G_Device_Health_Report confirm	
Neighbor_Health_Report		G_Neighbor_Health_Report request	
		G_Neighbor_Health_Report confirm	
Network_Health_Report		G_Network_Health_Report request	
		G_Network_Health_Report confirm	
Time		G_Time request	
		G_Time confirm	
Client/Server		G_Client_Server request	
		G_Client_Server indication	
		G_Client_Server response	
		G_Client_Server confirm	
Publish/Subscribe	Publish	G_Publish request	
		G_Publish indication	
		G_Publish confirm	
	Subscribe	G_Subscribe request	
		G_Subscribe confirm	
	Publish_Timer	G_Publish_Timer indication	
Bulk_Transfer <sup>1</sup>	Open	G_Bulk_Open request	Allows upload and download of large items such as firmware images and sample buffers
		G_Bulk_Open confirm	
	Transfer	G_Bulk_Transfer request	
		G_Bulk_Transfer confirm	
	Close	G_Bulk_Close request	
		G_Bulk_Close confirm	
Alert	Subscribe	G_Alert_Subscription request	Allows subscription and receipt of specific alerts
		G_Alert_Subscription confirm	
	Notify	G_Alert_Notification indication	
Gateway_Configuration	Read	G_Read_Gateway_Configuration request	
		G_Read_Gateway_Configuration confirm	
	Write	G_Write_Gateway_Configuration request	
		G_Write_Gateway_Configuration confirm	
Device_Configuration	Read	G_Read_Device_Configuration request	
		G_Read_Device_Configuration confirm	
	Write	G_Write_Device_Configuration request	
		G_Write_Device_Configuration confirm	

NOTE 1 The service primitives are common to both upload and download operations.

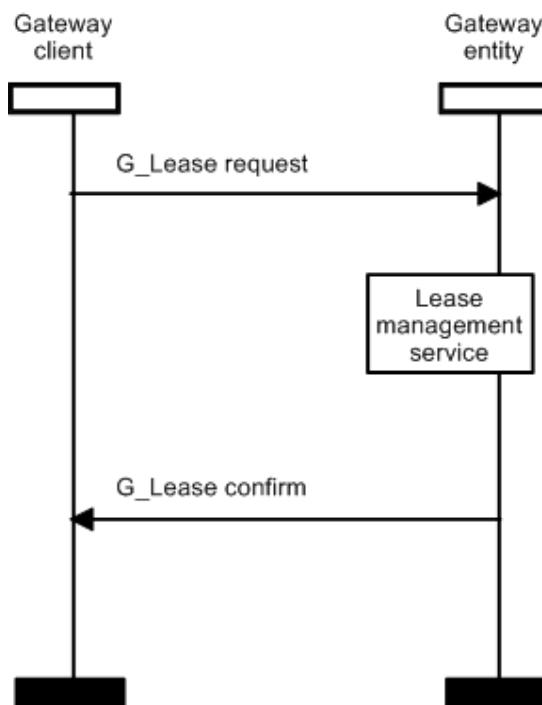
### 13.2.2 Sequence of primitives

Figure 137, Figure 138, Figure 139, Figure 140, Figure 141, Figure 142, Figure 143, Figure 144, Figure 145, Figure 146, Figure 147, Figure 148, and Figure 149 show the sequences of primitives for gateway high side services. The figures are defined in terms of a gateway client, a gateway entity, a device client, and a device entity. A gateway client is a user of the GSAP services within a gateway. A gateway entity is a provider of the GSAP services within the gateway. The provision of the services entails additional interactions across the wireless

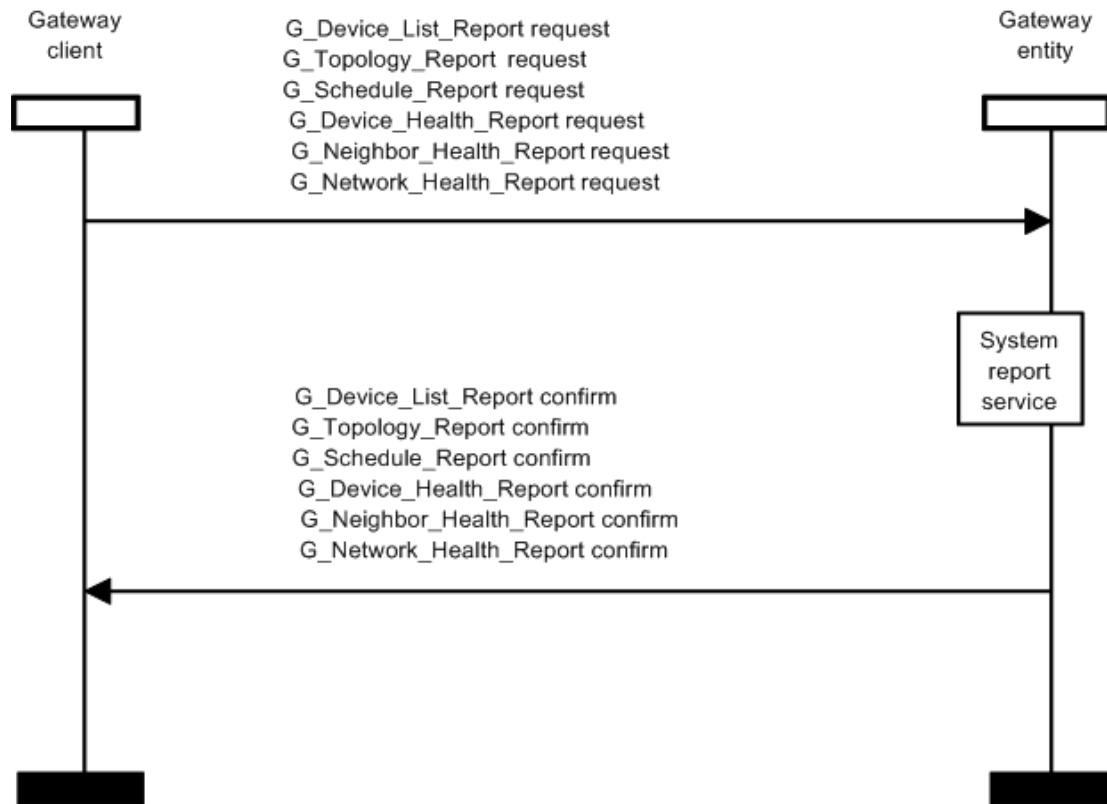
network to one or more devices. A device client is a user of the GSAP services within a device. A device entity is a provider of the GSAP services within the device.



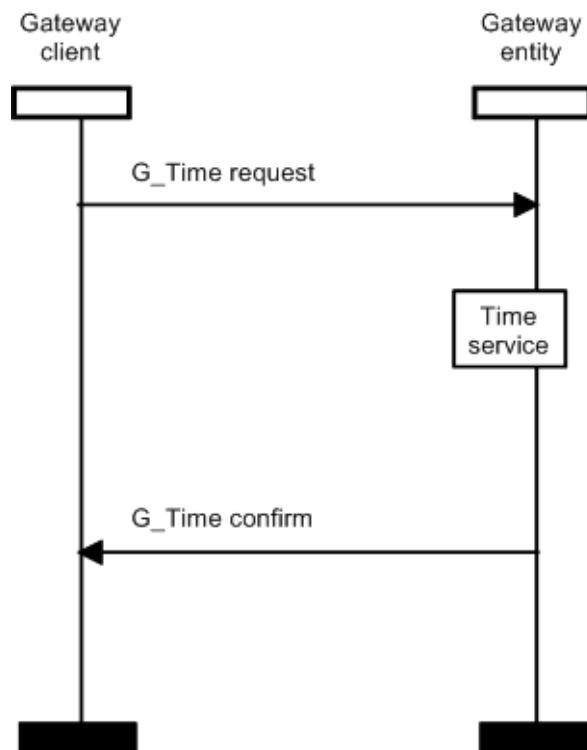
**Figure 137 – Sequence of primitives for session service**



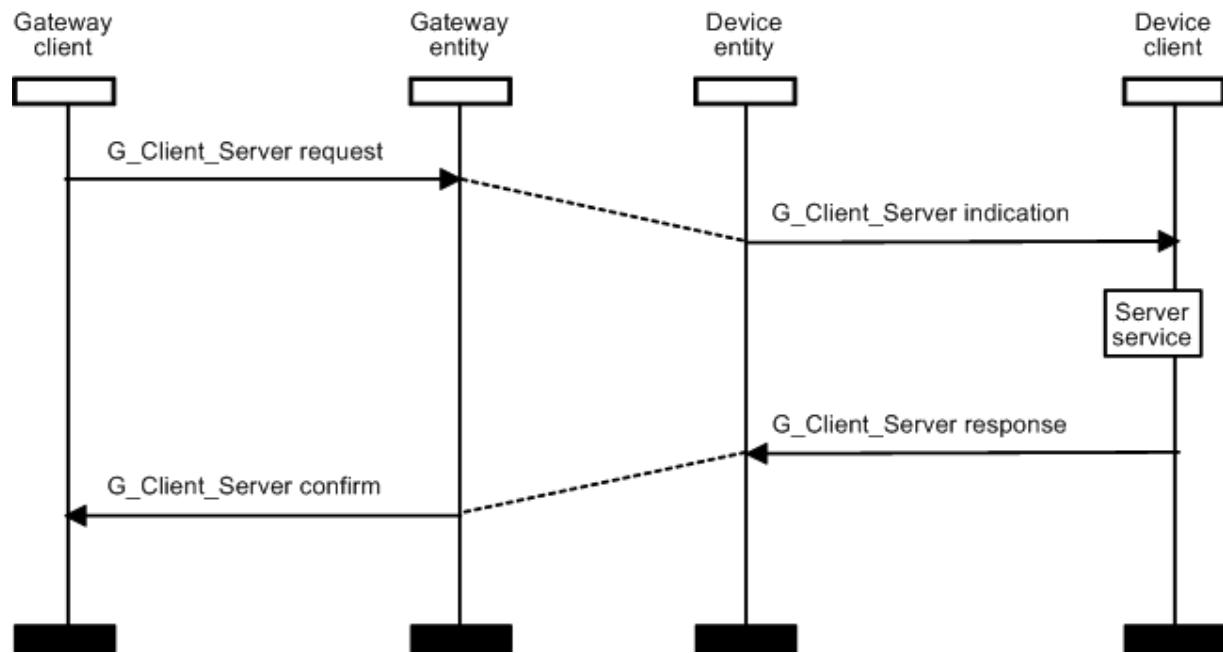
**Figure 138 – Sequence of primitives for lease management service**



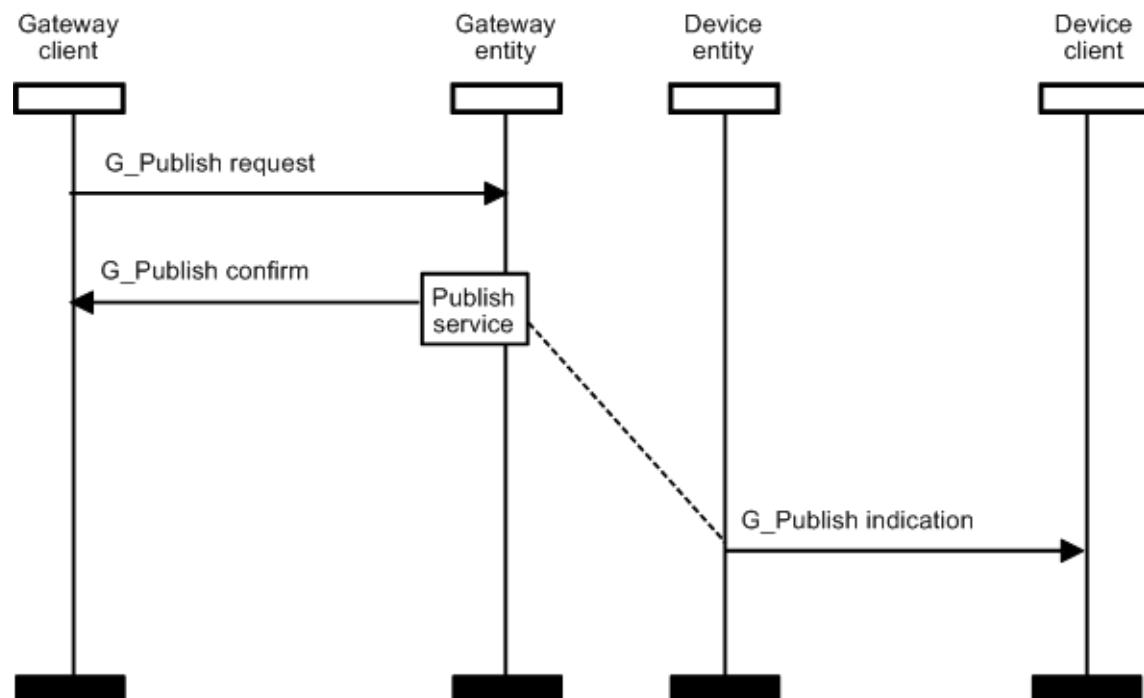
**Figure 139 – Sequence of primitives for system report services**



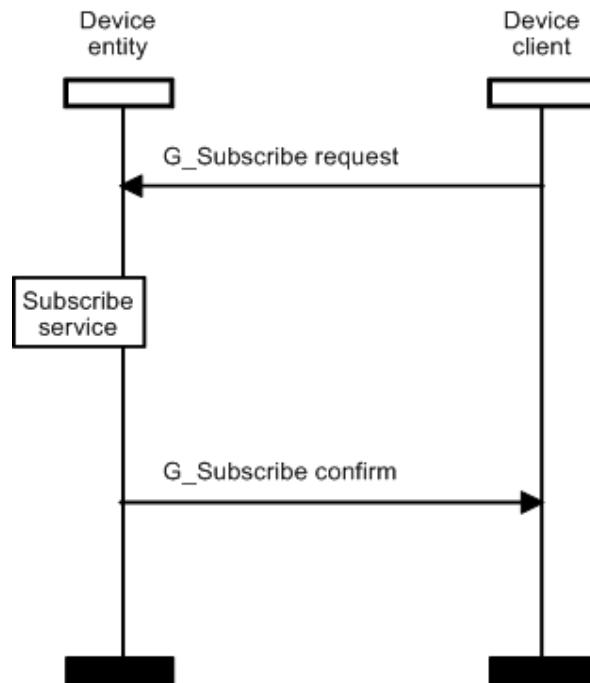
**Figure 140 – Sequence of primitives for time service**



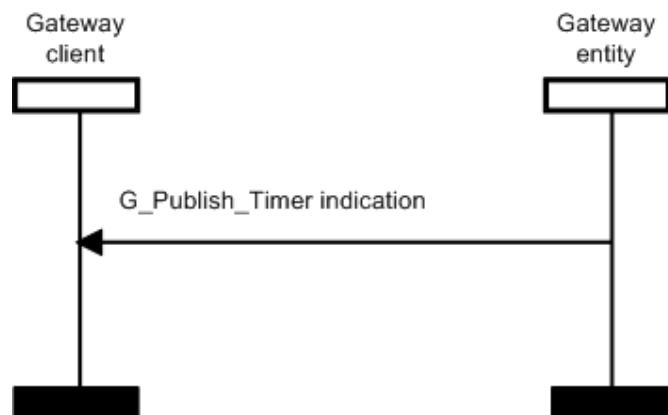
**Figure 141 – Sequence of primitives for client/server service initiated from gateway**



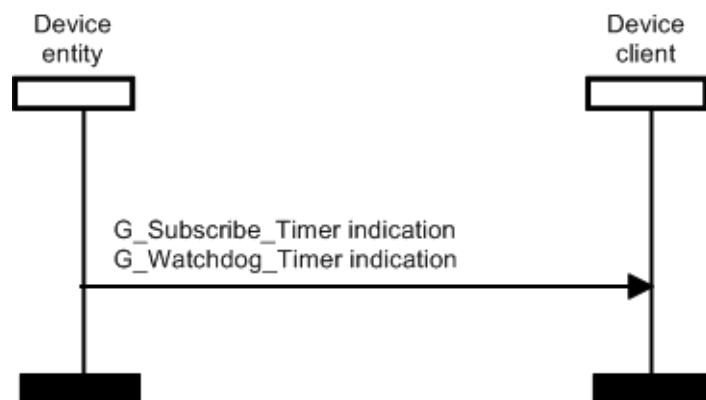
**Figure 142 – Sequence of primitives for publish service initiated from gateway**



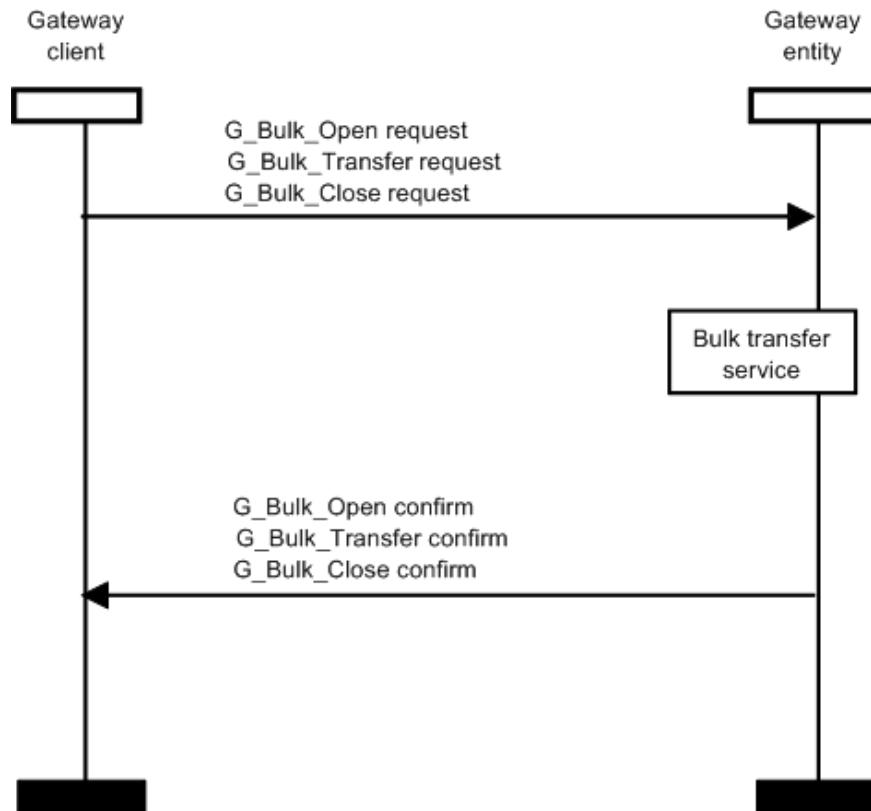
**Figure 143 – Sequence of primitives for subscribe service initiated from device**



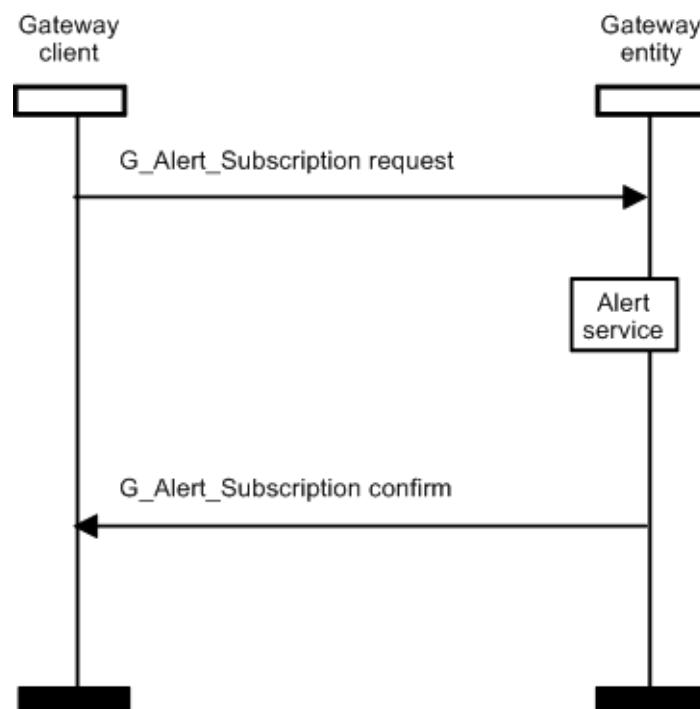
**Figure 144 – Sequence of primitives for publisher timer initiated from gateway**



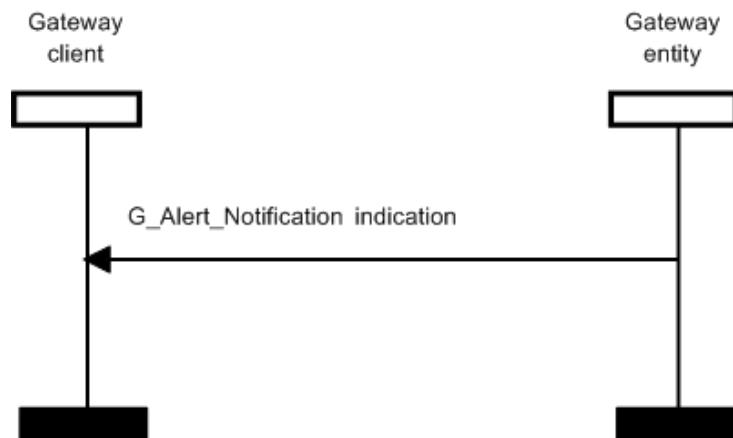
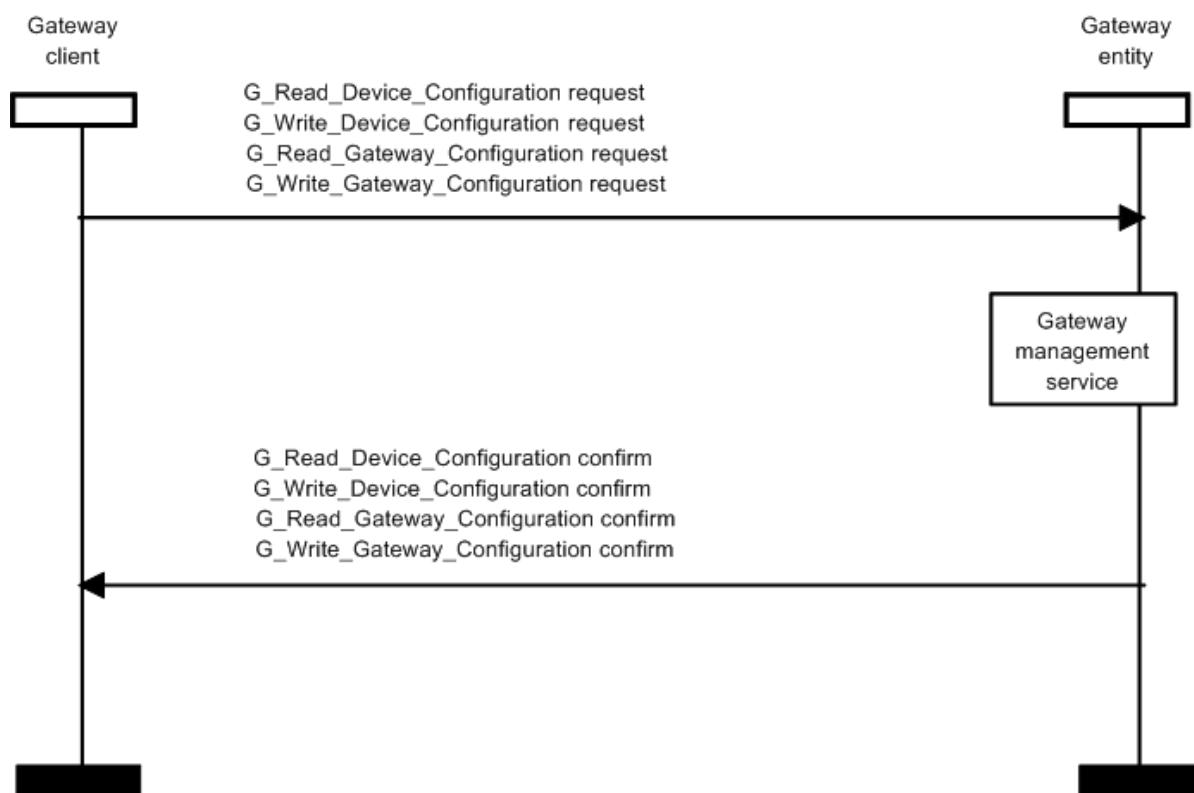
**Figure 145 – Sequence of primitives for subscriber timers initiated from device**



**Figure 146 – Sequence of primitives for the bulk transfer service**



**Figure 147 – Sequence of primitives for the alert subscription service**

**Figure 148 – Sequence of primitives for the alert notification service****Figure 149 – Sequence of primitives for gateway management services**

### 13.2.3 Detailed description of parameters

#### 13.2.3.1 General

Parameters that are common to multiple services are defined below. Parameters that are unique to a service are defined within that service.

#### 13.2.3.2 Parameter GS\_Session\_ID

##### 13.2.3.2.1 General

The parameter GS\_Session\_ID uniquely identifies a specific session.

##### 13.2.3.2.2 Use of the parameter

A valid session identifier that has not expired shall be provided in order to invoke all services except the session services.

### **13.2.3.3 Parameter GS\_Transaction\_ID**

#### **13.2.3.3.1 General**

The parameter GS\_Transaction\_ID uniquely identifies request and response portions of a transaction within the context of a specific session.

#### **13.2.3.3.2 Use of the parameter**

The transaction identifier shall be used to match a GSAP service request with the corresponding GSAP service response.

### **13.2.3.4 Parameter GS\_Lease\_ID**

#### **13.2.3.4.1 General**

GS\_Lease\_ID identifies gateway entity resources and communication resources that are allocated to a particular session.

#### **13.2.3.4.2 Use of the parameter**

The lease identifier shall be provided by the GSAP service user when a service is invoked to identify the particular communication resources utilized to support the service.

### **13.2.3.5 Parameter GS\_Status**

#### **13.2.3.5.1 General**

GS\_Status shall be returned by a confirm primitive. It may represent either the status resulting from handling a local request or the status corresponding to a response received from a remote entity.

#### **13.2.3.5.2 Use of the parameter**

The status shall indicate the success or failure of the service call and, when applicable, the reason for failure.

### **13.2.3.6 Parameter GS\_Network\_Address**

#### **13.2.3.6.1 General**

GS\_Network\_Address shall be a 128-bit address used to identify a logical device that is unique across all networks.

#### **13.2.3.6.2 Use of the parameter**

This parameter uniquely identifies a specific logical device for communication independent of the specific physical device. Communication and internal application configuration may use short addresses that are matched to a corresponding full length network address.

### **13.2.3.7 Parameter GS\_Uuid\_Device\_ID**

#### **13.2.3.7.1 General**

GS\_Uuid\_Device\_ID shall be a 64-bit device unique identifier in EUI-64 format.

#### **13.2.3.7.2 Use of the parameter**

This parameter uniquely identifies a specific physical device for asset management purposes.

### **13.2.3.8 Parameter GS\_Network\_ID**

#### **13.2.3.8.1 General**

GS\_Network\_ID shall be a unique identifier for one of several networks that may be accessible through a single gateway.

#### **13.2.3.8.2 Use of the parameter**

This parameter uniquely identifies a specific network.

### **13.2.3.9 Parameter GS\_Time**

#### **13.2.3.9.1 General**

GS\_Time shall be a 48-bit TAI time field.

#### **13.2.3.9.2 Use of the parameter**

The time parameter shall be used to specify time related fields such as timestamp, start time, and stop time.

### **13.2.3.10 Parameter GS\_Transfer\_Mode**

#### **13.2.3.10.1 General**

GS\_Transfer\_Mode identifies GPDUs-level transfer variations.

#### **13.2.3.10.2 Use of the parameter**

GS\_Transfer\_Mode shall be provided with each GPDUs provided for transfer in order to indicate the quality of service and the priority associated with the transfer of the GPDUs.

### **13.2.4 Detailed description of services**

#### **13.2.4.1 Session management service**

##### **13.2.4.1.1 General**

A gateway entity is a process within a gateway that provides gateway services through the GSAP. A gateway client is a user of gateway entity-provided services. Typical gateway clients include host systems, asset management systems, and engineering tools.

Gateway entities provide services to gateway clients within the context of a session. The session management service shall be used to establish and manage these sessions. All other gateway entity-provided services shall be used within the context of an established session.

A foreign protocol translator within the gateway may establish sessions on behalf of remote clients and perform protocol translation on the communication flows that correspond to gateway entity-provided services.

The primary purpose of a session is to allow resource allocation and bulk reclamation of gateway and communication resources on a per-gateway entity client basis.

A session may be established by a local process or remotely (such as through a TCP/IP remote session).

One or more sessions may exist concurrently between a gateway entity and one or more gateway clients. Each session shall be uniquely identified.

**NOTE** The number of concurrent sessions supported is implementation-dependent. It could be that some implementations will provide a fixed function gateway with a single session, while other implementations will provide a number of sessions that are allocated on demand to a variety of applications, including host systems, historians, asset management tools, and engineering tools.

##### **13.2.4.1.2 Use of the service**

The gateway client uses the G\_Session primitive to create, renew or delete a session.

##### **13.2.4.1.3 G\_Session primitive**

###### **13.2.4.1.3.1 General**

Table 372 specifies parameter usage for the primitive G\_Session.

**Table 372 – Primitive G\_Session parameter usage**

<b>Parameter name</b>	<b>G_Session</b>	
	<b>Request input</b>	<b>Confirm output</b>
GS_Session_ID	M	M
GS_Session_Period	M	M
GS_Network_ID	M	
GS_Status		M

#### 13.2.4.1.3.2 Use of G\_Session request

The gateway client uses the primitive G\_Session request to create, renew or delete a session.

A session shall be created by providing a null session identifier (GS\_Session\_ID = 0) and a requested session duration. Submitting GS\_Session\_Period > 0 requests a limited duration session, specified in seconds. Submitting GS\_Session\_Period = -1 requests an indefinite session duration.

A limited duration session shall be renewed by providing an existing (non-null) session identifier and session duration greater than 0 seconds. An indefinite duration session does not need to be renewed. A limited duration session cannot be changed to an indefinite duration session by renewal with duration of -1 seconds.

NOTE The upper bound of session duration is implementation-dependent and is not specified by this standard. It may be that certain implementations will dedicate resources to specific applications, such as a host system, and will never release the resources.

A session shall be deleted by providing an existing (non-null) session identifier and session duration of 0 seconds.

A gateway may connect to multiple networks. Each session shall be associated with a specific network. A network identifier (GS\_Network\_ID) shall be specified to establish a particular network for the session. The scope of further identifiers used within a session shall be limited to the particular network.

#### 13.2.4.1.3.3 Use of G\_Session confirm

The gateway entity uses the G\_Session confirm primitive to complete the G\_Session request to the gateway client.

For a successful session creation request, the gateway entity returns a unique, non-null session identifier. This identifier shall be used in subsequent session renew and delete operations. GS\_Session\_Period shall be returned with the actual session duration allocated by the gateway entity.

NOTE The GS\_Session\_Period value returned may not be the same as the value requested. How the value returned is determined is implementation-dependent and is not specified by this standard.

For a session renew request, the request session identifier shall be echoed, and a new session duration shall be returned.

For a session deletion request, the session identifier shall be echoed, and the session duration shall be set to 0 seconds.

GS\_Status shall be returned to indicate the success or failure of the operation, as described in Table 373.

**Table 373 – GS\_Status for G\_Session confirm**

<b>Value</b>	<b>Meaning</b>
0	Success, new session created, renewed or deleted
1	Success, new session created or renewed with reduced period
2	Failure; session does not exist to renew or delete
3	Failure; session cannot be created (no additional sessions available) This may occur, for example, if sessions have expired, but have not explicitly been deleted
4	Failure; other

### 13.2.4.2 Lease management service

#### 13.2.4.2.1 General

Gateway entities allocate communication resources to gateway clients via leases. The lease management service shall be used to establish and manage leases.

The primary purpose of a lease is to allow fine-grained communication resource allocation and reclamation on a per-session basis.

Resources may be separately allocated depending on communication needs. For example, client/server, publish/subscribe, bulk transfer, and alert subscription resources may be separately allocated.

One or more leases may exist concurrently between a gateway entity and one or more gateway clients. Each lease shall be uniquely identified within a session.

#### 13.2.4.2.2 Use of the service

The gateway client uses the G\_Lease primitive to create, renew or delete a lease.

#### 13.2.4.2.3 G\_Lease primitive

##### 13.2.4.2.3.1 General

Table 374 specifies parameter usage for the primitive G\_Lease.

**Table 374 – Primitive G\_Lease parameter usage**

<b>Parameter name</b>	<b>G_Lease</b>	
	<b>Request input</b>	<b>Confirm output</b>
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	M
GS_Lease_Period	M	M
GS_Lease_Type	M	
GS_Protocol_Type	M	
GS_Network_Address_List	C	
GS_Network_Address	C	
GS_Resource	C	
GS_Lease_Parameters	C	
GS_Transfer_Mode	C	
GS_Update_Policy	C	
GS_Period	C	
GS_Phase	C	
GS_Stale_Limit	C	
GS_Connection_Info	C	
GS_Wireless_Parameters	C	
GS_Status		M

##### 13.2.4.2.3.2 Use of G\_Lease request

The gateway client uses the primitive G\_Lease request to create, renew, or delete a lease.

A session identifier (GS\_Session\_ID) shall be included in the G\_Lease request primitives.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

A lease shall be created by providing a null lease identifier (GS\_Lease\_ID = 0) and a requested lease duration. Submitting GS\_Lease\_Period > 0 requests a limited duration lease, specified in seconds. Submitting GS\_Lease\_Period = 0 requests an indefinite lease duration.

A limited duration lease shall be renewed by providing an existing (non-null) lease identifier and lease duration greater than 0 seconds. An indefinite duration lease does not need to be renewed. A limited duration lease cannot be changed to an indefinite duration lease by renewal with duration of 0 seconds.

**NOTE** The maximum value which may be requested is implementation-dependent and is not specified by this standard. It may be that certain implementations will dedicate resources to specific applications, such as a host system, and will never release the resources.

A lease shall be deleted by providing an existing lease identifier and a requested lease duration of 0 seconds.

Different types of leases are available, as specified by GS\_Lease\_Type and as shown in Table 375. Each lease type allocates lease-specific gateway entity resources and communication resources on behalf of the gateway client.

**Table 375 – GS\_Lease\_Type for G\_Lease request**

Value	Meaning
0	Client
1	Server
2	Publisher
3	Subscriber
4	Bulk transfer client
5	Bulk transfer server
6	Alert subscription

GS\_Protocol\_Type identifies the protocol that is associated with the lease, as indicated in Annex M. Specification of the protocol type allows special processing for particular protocols within the gateway entity.

All leases relate to establishing communication services between the gateway entity and one or more device specified by one or more elements in GS\_Network\_Address\_List. Alert subscription leases do not allocate communication resources during lease establishment, but dynamically as alert subscriptions are modified. GS\_Network\_Address\_List is not used with the alert subscription lease type.

Client, server, subscriber, bulk transfer client, and bulk transfer server shall only specify a single element within GS\_Network\_Address\_List. The publisher lease type may specify multiple elements.

The specification of multiple network addresses within the GS\_Network\_Address\_List represents a multicast group. Elements within the G\_Network\_Address\_List include GS\_Network\_Address.

GS\_Lease\_Parameters is a parameter structure for the specification of parameters necessary for the establishment of certain lease types. Usage of the GS\_Lease\_Parameters is conditioned on the specific lease type as follows.

**NOTE** Additional information on detailed GS\_Lease\_Parameters usage may be found in Annex P.

Client, server, publish, and subscribe leases shall specify a unique GS\_Resource. This value identifies matching client and server connection endpoints, and also identifies matching publisher and subscriber endpoints. GS\_Resource shall also be specified by the bulk transfer client to identify the upload/download item.

Publisher leases require specification of GS\_Update\_Policy, GS\_Period, GS\_Phase, and GS\_Stale\_Limit to control timing and buffered behavior. GS\_Transfer\_Mode shall also be specified in order to set default transfer quality of service and priority.

Subscriber leases require specification of GS\_Update\_Policy, GS\_Period, GS\_Phase, and GS\_Stale\_Limit to control timing and buffered behavior.

A publisher and subscriber may optionally agree to specify GS\_Connection\_Info in the subscriber lease for provision on each publication receipt.

Client and server leases shall specify GS\_Transfer\_Mode in order to set default transfer quality of service and priority.

An additional GS\_Wireless\_Parameters field usage depends on gateway construction. This allows access to all exposed, requestable communication features.

#### **13.2.4.2.3.3 Use of G\_Lease confirm**

The gateway entity uses the primitive G\_Lease confirm to complete the G\_Lease request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm primitive with the original request primitive.

For a successful lease create request, the gateway entity returns a session unique lease identifier. This lease identifier shall then be used in subsequent lease renew and delete operations. GS\_Lease\_Period is returned with the actual lease duration allocated by the gateway entity.

For a lease renew request, the request lease identifier shall be echoed, and the actual lease duration shall be given.

For a lease delete request, the lease identifier shall be echoed, and lease duration shall be set to 0 seconds.

GS\_Status shall be returned to indicate success or failure of the operation, as described in Table 376.

**Table 376 – GS\_Status for G\_Lease confirm**

Value	Meaning
0	Success, new lease created, renewed or deleted
1	Success, new lease created or renewed with reduced period
2	Failure; lease does not exist to renew or delete
3	Failure; no additional leases available
4	Failure; no device exists at network address
5	Failure; invalid lease type
6	Failure; invalid lease type information
7	Failure; other

#### **13.2.4.3 Device list report service**

##### **13.2.4.3.1 General**

The device list report service shall provide a report of the devices that are associated with a gateway. This is useful for mapping wireless devices to host systems and network browsers.

##### **13.2.4.3.2 Use of the service**

The gateway client shall use the G\_Device\_List\_Report primitive to retrieve a report on the devices associated with a gateway entity.

### 13.2.4.3.3 G\_Device\_List\_Report primitive

#### 13.2.4.3.3.1 General

Table 377 specifies parameter usage for the primitive G\_Device\_List\_Report.

**Table 377 – Primitive G\_Device\_List\_Report parameter usage**

<b>Parameter name</b>	<b>G_Device_List_Report</b>	
	<b>Request input</b>	<b>Confirm output</b>
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List		M
GS_Network_Address		M
GS_Device_Type		M
GS_Unique_Device_ID		M
GS_Manufacturer		M
GS_Model		M
GS_Revision		M
GS_Status		M

#### 13.2.4.3.3.2 Use of G\_Device\_List\_Report request

The gateway client uses the primitive G\_Device\_List\_Report request to retrieve a report on the devices associated with a gateway entity.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

#### 13.2.4.3.3.3 Use of G\_Device\_List\_Report confirm

The gateway entity uses the primitive G\_Device\_List\_Report confirm to complete the G\_Device\_List\_Report request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm with the original request.

A list of devices associated with the gateway entity (GS\_Device\_List) shall be returned. For each device, the list shall include the network address (GS\_Network\_Address), the type of the device (GS\_Device\_Type), and the unique device identifier (GS\_Unique\_Device\_ID).

The list also includes additional manufacturer related information (GS\_Manufacturer, GS\_Model, and GS\_Revision).

Where the gateway includes the role of provisioning device, the network address may be a default address. The unique identifier and the manufacturer information are utilized within the host-level applications to control device commissioning through the device configuration service.

For example, a browser may display a list of devices available for provisioning along with identification information. A select set of the devices picked from the display and are commissioned with network address and other information to join the system. The browsing display is then refreshed with devices that are available for linkage to a control strategy.

GS\_Status shall be returned to indicate success or failure of the operation, as described in Table 378.

**Table 378 – GS\_Status for G\_Device\_List\_Report confirm**

Value	Meaning
0	Success
1	Failure

### 13.2.4.4 Topology report service

#### 13.2.4.4.1 General

The topology report service provides a topology report that relates devices within a wireless mesh.

#### 13.2.4.4.2 Use of the service

The gateway client uses the G\_Topo\_Report primitive to retrieve a report on the devices associated with a gateway entity.

#### 13.2.4.4.3 G\_Topo\_Report primitive

##### 13.2.4.4.3.1 General

Table 379 specifies parameter usage for the primitive G\_Topo\_Report.

**Table 379 – Primitive G\_Topo\_Report parameter usage**

Parameter name	G_Topo_Report	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List		M
GS_Network_Address		M
GS_Neighbor_List		M
GS_Network_Address		M
GS_Graph_List		M
GS_Graph_ID		M
GS_Network_Address		M
GS_Status		M

#### 13.2.4.4.3.2 Use of G\_Topo\_Report request

The gateway client uses the primitive G\_Topo\_Report request to retrieve a report on the topology of the devices associated with a gateway entity.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

#### 13.2.4.4.3.3 Use of G\_Topo\_Report confirm

The gateway entity uses the primitive G\_Topo\_Report confirm to complete the G\_Topo\_Report request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) shall be returned to allow matching of a confirm with the original request.

A list of devices associated with the gateway entity (GS\_Device\_List) shall be returned. The list includes the network address (GS\_Network\_Address) for each device.

Also included within the list shall be a second list (GS\_Neighbor\_List) of the neighbor devices associated with each device. The list includes the network address (GS\_Network\_Address) of all neighbors (as defined in the neighbor health report).

Also included within the list shall be a third list (GS\_Graph\_List) of the graph connections associated with each device. For each graph connection, the list shall include the graph identified (GS\_Graph\_ID) and an associated network address list (GS\_Network\_Address) of the neighbors on the graph.

GS\_Status shall be returned to indicate success or failure of the operation, as described in Table 380.

**Table 380 – GS\_Status for G\_Topoogy\_Report confirm**

Value	Meaning
0	Success
1	Failure

### 13.2.4.5 Schedule report service

#### 13.2.4.5.1 General

The schedule report service provides a schedule report detailing time slot and channel allocations on a per-device basis.

#### 13.2.4.5.2 Use of the service

The gateway client uses the G\_Schedule\_Report primitive to retrieve a report on the schedule of the devices associated with a gateway entity.

#### 13.2.4.5.3 G\_Schedule\_Report primitive

##### 13.2.4.5.3.1 General

Table 381 specifies parameter usage for the primitive G\_Schedule\_Report.

**Table 381 – Primitive G\_Schedule\_Report parameter usage**

Parameter name	G_Schedule_Report	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Network_Address	M	
GS_Channel_List		M
GS_Channel_Number		M
GS_Channel_Status		M
GS_Device_Schedule		M
GS_Network_Address		M
GS_Superframe_List		C
GS_Superframe_ID		C
GS_Num_Time_Slots		C
GS_Start_Time		C
GS_Link_List		C
GS_Network_Address		C
GS_Slot_Length		C
GS_Channel		C
GS_Direction		C
GS_Link_Type		C
GS_Status		M

#### 13.2.4.5.3.2 Use of G\_Schedule\_Report request

The gateway client uses the primitive G\_Schedule\_Report request to retrieve a schedule report for a specific device associated with a gateway entity. The particular device is identified by its network address (GS\_Network\_Address).

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

#### 13.2.4.5.3.3 Use of G\_Schedule\_Report confirm

The gateway entity uses the primitive G\_Schedule\_Report confirm to complete the G\_Schedule\_Report request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm with the original request.

A list of channels is returned. Each element of the list includes the channel number (GS\_Channel\_Number) and the status of the channel (GS\_Channel\_Status). Channel status is set to 0 to indicate a disabled channel or 1 to indicate an enabled channel.

A device schedule (GS\_Device\_Schedule) is also returned. The schedule includes device identification information (GS\_Network\_Address) and a list of superframes (GS\_Superframe\_List) that are used by the device for communication.

If a device does not use superframes, GS\_Superframe\_List is not returned.

The superframe list includes general superframe information, including a superframe identifier (GS\_Superframe\_ID), the number of time slots in the frame (GS\_Num\_Time\_Slots), and the start time of the superframe (GS\_Start\_Time). Only active superframes are reported.

GS\_Start\_Time is an offset relative to the beginning of TAI time. This number has significance only relative to the current network time, unless the communication is synchronized to an external source. GS\_Start\_Time shall be set to -1 to indicate that the superframe has no known synchronization.

The superframe list also includes an ordered list (GS\_Link\_List) with one element per timeslot in the superframe. The link list elements are used to specify communication relationships related to the superframe. Each link list element specifies the timeslot length in microseconds (GS\_Slot\_Length) and the channel (GS\_Channel) for communication within the superframe. The link options (GS\_Direction) parameter specifies the direction of communications. A value of 0 specifies reception, and a value of 1 specifies transmission. The network address (GS\_Network\_Address) specifies the logical network address of a communication partner for the slot.

The link type (GS\_Link\_Type) specifies the purpose of the communication:

- A value of 0 specifies aperiodic data communication.
- A value of 1 specifies aperiodic management communication.
- A value of 2 specifies periodic data communication.
- A value of 3 specifies periodic management communication.

GS\_Status is returned to indicate success or failure of the operation, as described in Table 382.

**Table 382 – GS\_Status for G\_Schedule\_Report confirm**

Value	Meaning
0	Success
1	Failure

### 13.2.4.6 Device health report service

#### 13.2.4.6.1 General

The device health report service provides a communication health report for each device's view of its own health.

#### 13.2.4.6.2 Use of the service

The gateway client uses the G\_Device\_Health\_Report primitive to retrieve a device health report for a specified set of devices that are associated with a gateway entity.

#### 13.2.4.6.3 G\_Device\_Health\_Report primitive

##### 13.2.4.6.3.1 General

Table 383 specifies parameter usage for the primitive G\_Device\_Health\_Report.

**Table 383 – Primitive G\_Device\_Health\_Report parameter usage**

Parameter name	G_Device_Health_Report	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List	M	M
GS_Network_Address	M	M(=)
GS_DPDUs_Transmitted		M
GS_DPDUs_Received		M
GS_DPDUs_Failed_Transmission		M
GS_DPDUs_Failed_Reception		M
GS_Status		M

##### 13.2.4.6.3.2 Use of G\_Device\_Health\_Report request

The gateway client uses the primitive G\_Device\_Health\_Report request to retrieve a device health report for a specified set of devices that are associated with a gateway entity.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

A health report is requested for a specific list of devices (GS\_Device\_List). The network address of each device (GS\_Network\_Address) is required.

##### 13.2.4.6.3.3 Use of G\_Device\_Health\_Report confirm

The gateway entity uses the primitive G\_Device\_Health\_Report confirm to complete the G\_Device\_Health\_Report request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm with the original request.

A device list (GS\_Device\_List) is returned. The list includes device identification information (GS\_Network\_Address) and communication health information. The communication health information includes the total number of DPDUs transmitted (GS\_DPDUs\_Transmitted) from the device to all neighbors, the total number of DPDUs received (GS\_DPDUs\_Received) from

the device by all neighbors, the total number of DPDUs to all neighbors that failed transmission (GS\_DPDUs\_Failed\_Transmission), and the total number of DPDUs from all neighbors that failed reception (GS\_DPDUs\_Failed\_Reception). Failed receptions shall include identifiable DPDUs that are discarded due to transmission related corruption.

NOTE Failed receptions will likely be less than failed transmissions, since many failed DPDUs will not have enough uncorrupted information to determine the addressing. Failed reception does not include protocol-related errors.

GS\_Status is returned to indicate success or failure of the operation, as described in Table 384.

**Table 384 – GS\_Status for G\_Device\_Health\_Report confirm**

Value	Meaning
0	Success
1	Failure

### 13.2.4.7 Neighbor health report service

#### 13.2.4.7.1 General

The neighbor health report service provides a communication health report for each device's view of its neighbors.

A neighbor device is a link level wireless communication partner that is configured for direct exchange of DPDUs (RF transmission without hops). The neighbor health report services provide information on these physical neighbors. Neighbor devices are able to collect DPDUs exchange statistics that indicate local RF conditions.

#### 13.2.4.7.2 Use of the service

The gateway client uses the G\_Neighbor\_Health\_Report primitive to retrieve a communication health report for the set of neighbor devices associated with a specific device that is associated with a gateway entity.

#### 13.2.4.7.3 G\_Neighbor\_Health\_Report primitive

##### 13.2.4.7.3.1 General

Table 385 specifies parameter usage for the primitive G\_Neighbor\_Health\_Report.

**Table 385 – Primitive G\_Neighbor\_Health\_Report parameter usage**

Parameter name	G_Neighbor_Health_Report	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Network_Address	M	
GS_Neighbor_Health_List		M
GS_Network_Address		M
GS_Link_Status		M
GS_DPDUs_Transmitted		M
GS_DPDUs_Received		M
GS_DPDUs_Failed_Transmission		M
GS_DPDUs_Failed_Reception		M
GS_Signal_Strength		M
GS_Signal_Quality		M
GS_Status		M

### **13.2.4.7.3.2 Use of G\_Neighbor\_Health\_Report request**

The gateway client uses the primitive G\_Neighbor\_Health\_Report request to retrieve a communication health report for the set of neighbor devices associated with a specific device that is associated with a gateway entity.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

A neighbor health report is requested for a device at a specific network address (GS\_Network\_Address).

### **13.2.4.7.3.3 Use of G\_Neighbor\_Health\_Report confirm**

The gateway entity uses the primitive G\_Neighbor\_Health\_Report confirm to complete the G\_Neighbor\_Health\_Report request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm with the original request.

A neighbor health list (GS\_Neighbor\_Health\_List) is returned. The list includes the neighbor device identification information (GS\_Network\_Address) and communication health information. The communication health information includes a general status (GS\_Link\_Status). GS\_Link\_Status = 1 indicates that the neighbor is available for communication. GS\_Link\_Status = 0 indicates that the neighbor is unavailable for communication.

Health information also includes the number of DPDUs transmitted to the neighbor (GS\_DPDUs\_Transmitted), the number of DPDUs received from the neighbor (GS\_DPDUs\_Received), the number of failed transmission attempts (GS\_DPDUs\_Failed\_Transmission), and the number of failed receptions (GS\_DPDUs\_Failed\_Reception) from the neighbor. Failed receptions shall include identifiable DPDUs that are discarded due to transmission related corruption.

NOTE Failed receptions will likely be less than failed transmissions, since many failed DPDUs will not have enough uncorrupted information to determine the addressing. Failed reception does not include protocol-related errors.

Health information also includes GS\_Signal\_Strength and GS\_Signal\_Quality. These parameters return values between 0 (worst signal) and 100 (best signal). GS\_Signal\_Strength indicates the average uncorrelated power level of the signals received from a specific neighbor relative to the range of the receiver. GS\_Signal\_Quality indicates the average correlated power level of the signals received from a specific neighbor relative to the range of the receiver.

GS\_Status is returned to indicate success or failure of the operation, as described in Table 386.

**Table 386 – GS\_Status for G\_Device\_Health\_Report confirm**

Value	Meaning
0	Success
1	Failure

## **13.2.4.8 Network health report service**

### **13.2.4.8.1 General**

The network health report service provides a summary communication health report for an entire network.

### 13.2.4.8.2 Use of the service

The gateway client uses the G\_Network\_Health\_Report primitive to retrieve a summary communication health report for an entire network.

### 13.2.4.8.3 G\_Network\_Health\_Report primitive

#### 13.2.4.8.3.1 General

Table 387 specifies parameter usage for the primitive G\_Network\_Health\_Report.

**Table 387 – Primitive G\_Network\_Health\_Report parameter usage**

<b>Parameter name</b>	<b>G_Network_Health_Report</b>	
	<b>Request input</b>	<b>Confirm output</b>
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Network_Health		M
GS_Network_ID		M
GS_Network_Type		M
GS_Device_Count		M
GS_Start_Date		M
GS_Current_Date		M
GS_DPDUs_Sent		M
GS_DPDUs_Lost		M
GS_GPDU_Latency		M
GS_GPDU_Path_Reliability		M
GS_GPDU_Data_Reliability		M
GS_Join_Count		M
GS_Device_Health_List		M
GS_Network_Address		M
GS_Start_Date		M
GS_Current_Date		M
GS_DPDUs_Sent		M
GS_DPDUs_Lost		M
GS_GPDU_Latency		M
GS_GPDU_Path_Reliability		M
GS_GPDU_Data_Reliability		M
GS_Join_Count		M
GS_Status		M

#### 13.2.4.8.3.2 Use of G\_Network\_Health\_Report request

The gateway client uses the primitive G\_Network\_Health\_Report request to retrieve a summary communication health report for an entire network.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

#### 13.2.4.8.3.3 Use of G\_Network\_Health\_Report confirm

The gateway entity uses the primitive G\_Network\_Health\_Report confirm to complete the G\_Network\_Health\_Report request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm with the original request.

A network health summary (GS\_Network\_Health) is returned. The summary includes network identification information (GS\_Network\_ID and GS\_Network\_Type) and network communication health summary information. The communication health information includes

the number of devices in the network (GS\_Device\_Count), the start date and the current date for the network (GS\_Start\_Date and GS\_Current\_Date), transmission statistics (GS\_DPDUs\_Sent, GS\_DPDUs\_Lost, and GS\_GPDU\_Latency), reliability statistics (GS\_GPDU\_Path\_Reliability and GS\_GPDU\_Data\_Reliability), and join statistics (GS\_Join\_Count).

A device-specific health summary (GS\_Device\_Health\_List) is also returned. The list includes device identification information (GS\_Network\_Address) and communication statistics that are an identical subset of those contained in the network health summary (GS\_Start\_Date, GS\_Current\_Date, GS\_DPDUs\_Sent, GS\_DPDUs\_Lost, GS\_GPDU\_Latency, GS(gpdu)\_Path\_Reliability, GS(gpdu)\_Data\_Reliability, and GS\_Join\_Count).

GS\_Start\_Date is a 48-bit TAI time field indicating the time when a device first started operating. This is useful for calculation of battery replacement schedules.

GS\_Current\_Date is a 48-bit TAI time field indicating the current time as viewed by the device. This is the time used by the device for timestamp purposes.

GS\_GPDU\_Latency is a number from 0 to 100 indicating the percentage of scheduled GPDUs that arrive later than expected. These GPDUs may be delayed due to delivery over secondary paths or due to congestion in intermediate devices.

GS\_GPDU\_Path\_Reliability is a number from 0 to 100 indicating the percentage of first path success for acknowledged GPDU transmission. GPDUs that are transmitted on a secondary path may arrive successfully, but may reduce the path reliability.

GS\_GPDU\_Data\_Reliability is a number from 0 to 100 indicating the percentage of total GPDUs that are successful GPDUs. The total GPDUs is the number of acknowledged transmit GPDUs that are attempted plus the number of received GPDUs. Successful GPDUs are acknowledged transmit GPDUs that are transferred correctly on the first attempt plus receive GPDUs that pass integrity checks.

GS\_Join\_Count is a positive integer that indicates the number of times a device has joined the system. Join count may rise if power is interrupted, a device is reset, the network is reformed, or a device is moved to a new network. Excessive joins may indicate device integrity or communication problems.

GS\_Status is returned to indicate success or failure of the operation, as described in Table 388.

**Table 388 – GS\_Status for G\_Network\_Health\_Report confirm**

Value	Meaning
0	Success
1	Failure

### 13.2.4.9 Time service

#### 13.2.4.9.1 General

The time service enables retrieval and setting of the time for a wireless network associated with a gateway. This is useful for time synchronization of a network of wireless devices with a host system and other host-level applications.

#### 13.2.4.9.2 Use of the service

The gateway client shall use the G\_Time primitive to retrieve a report on the devices associated with a gateway entity.

#### 13.2.4.9.3 G\_Time primitive

##### 13.2.4.9.3.1 General

Table 389 specifies parameter usage for the primitive G\_Time.

**Table 389 – Primitive G\_Time parameter usage**

<b>Parameter name</b>	<b>G_Time</b>	
	<b>Request input</b>	<b>Confirm output</b>
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Command	M	
GS_Time	C	M
GS_Status		M

**13.2.4.9.3.2 Use of G\_Time request**

The gateway client uses the primitive G\_Time request to read or set the network time.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

GS\_Command = 0 shall read the network time. GS\_Time shall not be included.

GS\_Command = 1 shall attempt to set the network time. A new time (GS\_Time) shall be provided.

**13.2.4.9.3.3 Use of G\_Time confirm**

The gateway entity uses the primitive G\_Time confirm to complete the G\_Time request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm with the original request.

If GS\_Command = 0 in the request, the current network time (GS\_Time) shall be returned.

If GS\_Command = 1 in the request, the service shall attempt to set the network time. The current network time (GS\_Time) shall be returned. If the update is successful, the current time will reflect the change.

GS\_Status shall be returned to indicate success or failure of the operation, as described in Table 390.

**Table 390 – GS\_Status for G\_Time confirm**

<b>Value</b>	<b>Meaning</b>
0	Success
1	Failure; not allowed to set time in this configuration
2	Failure; other

**13.2.4.10 Client/server service****13.2.4.10.1 General**

The client/server service shall provide for client/server data transfer. The necessary communication resources to enable the transfer are allocated through the use of the lease service. The client and server each perform separate but related roles. Linkage of the client and the server shall be accomplished through the establishment of leases with matching lease information. Communication resources shall include local buffer facilities in order to minimize energy consuming transactions. Clients and servers may exist either in the gateway or in devices.

#### 13.2.4.10.2 Use of the service

The G\_Client\_Server primitive shall be used to send a client request data payload to a server and to initiate receipt of a corresponding server response data payload. The response payload may come from the client buffer or from the server.

#### 13.2.4.10.3 G\_Client\_Server primitive

##### 13.2.4.10.3.1 General

Table 391 specifies parameter usage for the primitive G\_Client\_Server.

**Table 391 – Primitive G\_Client\_Server parameter usage**

Parameter name	G_Client_Server			
	Request input	Indication output	Response input	Confirm output
GS_Session_ID	M			M(=)
GS_Transaction_ID	M			M(=)
GS_Lease_ID	M			
GS_Buffer	M			
GS_Transfer_Mode	M		M	
GS_Request_Data	M	C(=)		
GS_Response_Data			C	M
GS_Transaction_Info	C			C(=)
GS_Status			M	M

##### 13.2.4.10.3.2 Use of G\_Client\_Server request

The primitive G\_Client\_Server request shall be used to send a client request data payload (GS\_Request\_Data) to a server and to initiate receipt of a corresponding server response data payload (GS\_Response\_Data).

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

The server device is known through the lease identifier (GS\_Lease\_ID) that was obtained from the lease service.

The response data will be requested from the server device if the buffer is disabled for the transaction (GS\_Buffer = 0). The response data will be delivered from the buffer if the buffer is enabled for the transaction (GS\_Buffer = 1) and the buffer contains a matching response that has not expired.

GS\_Transfer\_Mode shall be provided with the request in order to indicate the quality of service and priority for the transfer of the data.

GS\_Transaction\_Info may optionally be provided in a request. The same information shall be returned in a confirm.

##### 13.2.4.10.3.3 Use of G\_Client\_Server indication

The primitive G\_Client\_Server indication shall be used to signal the arrival of a client request data payload at the server for processing.

The indication shall be conditional on whether the server response data payload could be delivered from the client buffer.

##### 13.2.4.10.3.4 Use of G\_Client\_Server response

The primitive G\_Client\_Server response shall be used to return a server response data payload to the client.

GS\_Transfer\_Mode shall be provided with the response in order to indicate the quality of service and priority for the transfer of the data.

The response shall be conditional on whether the server response data payload could be delivered from the client buffer.

#### **13.2.4.10.3.5 Use of G\_Client\_Server confirm**

The primitive G\_Client\_Server confirm shall be used to complete the G\_Client\_Server request to the client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm primitive with the original request primitive.

A server response data payload shall be returned. The payload is either delivered from the client buffer or from the server.

If GS\_Transaction\_Info was provided in the request, it will be returned in the confirm.

GS\_Status is returned to indicate success or failure of the operation, as described in Table 392.

**Table 392 – GS\_Status for G\_Client\_Server confirm**

Value	Meaning
0	Success
1	Failure; server is inaccessible for unbuffered request
2	Failure; server is inaccessible and client buffer is invalid for buffered request
3	Failure; lease has expired
4	Failure; other

### **13.2.4.11 Publish/subscribe service**

#### **13.2.4.11.1 General**

The publish/subscribe service shall provide mechanisms for publish/subscribe data transfer. The necessary communication resources to enable message exchange are allocated through the use of the lease service. The publisher and the subscriber each perform separate but related roles. Linkage of the publisher and the subscriber is accomplished through separate establishment of matching communication. Communication resources include local buffer facilities in both the publisher and the subscriber in order to minimize energy consuming transactions. Publishers and subscribers may exist in gateways, adapters, or native devices.

#### **13.2.4.11.2 Use of the service**

##### **13.2.4.11.2.1 Lease establishment**

The G\_Lease service shall be used prior to the use of the G\_Publish service in order to establish a GS\_Lease\_ID. The GS\_Lease\_Type shall be set to either publisher or subscriber to configure the respective side and to establish and reserve the underlying gateway entity and communication channel resources.

GS\_Network\_Address\_List shall be used by the publisher and subscriber to establish the identity of the other endpoints. A publisher may specify multiple addresses within the list in order to configure multiple subscribers.

Within the lease, GS\_Protocol\_Type shall be used to specify the application protocol that will be tunneled through the service. This allows protocol-specific processing to occur.

GS\_Lease\_Parameters shall be used to establish the expected protocol interaction between a publisher and a subscriber.

##### **13.2.4.11.2.2 Publication**

The G\_Publish primitive shall be used by a publisher to initiate transfer of a publish data payload to one or more subscribers.

The publish data payload is stored in a local buffer and forwarded from the buffer to subscribers. The lease configuration options determine when forwarding will occur. Forwarding occurs in order to meet scheduled deadlines. Over a period of time, the same payload may be forwarded multiple times to indicate that the publisher still exists and to prevent timeout. Invocation with unchanged data may not result in forwarding.

#### **13.2.4.11.2.3 Subscription**

The G\_Subscribe primitive shall be used by a subscriber to retrieve the most recent publication data from the local buffer.

The subscriber also receives the most recent publication associated with a subscribe lease via the G\_Publish indication primitive.

The primitive G\_Publish\_Watchdog shall be used within a subscriber to signal the expiration of a watchdog timer. The timer expires in the absence of expected updates from a publisher. The timer is reset on the arrival of publication data payload at the subscriber. The watchdog timer is configured with the lease configuration options.

The primitive G\_Publish\_Timer shall be used within a publisher to signal the expiration of a publication timer. The publication timer is a periodic timer that expires prior to the deadline for forwarding the publish data payload. The indication may be used to publish fresh data. The publication timer is configured with the lease configuration options.

The primitive G\_Subscribe\_Timer shall be used within a subscriber to signal the expiration of a subscription timer. The subscription timer shall be a periodic timer that expires at the delivery deadline for receiving the publish data payload. The indication shall be used to process existing publication data. Arrival of fresh data will reset the timer and will result in a G\_Publish indication. The subscription timer shall be configured with the lease configuration options.

#### **13.2.4.11.3 Types of primitives and parameters**

##### **13.2.4.11.3.1 G\_Publish primitive**

###### **13.2.4.11.3.1.1 General**

Table 393 specifies parameter usage for the primitive G\_Publish.

**Table 393 – Primitive G\_Publish parameter usage**

<b>Parameter name</b>	<b>G_Publish</b>		
	<b>Request input</b>	<b>Indication output</b>	<b>Confirm output</b>
GS_Session_ID	M	M	M
GS_Transaction_ID	M		M(=)
GS_Lease_ID	M	M	
GS_Transfer_Mode	M		
GS_Publish_Data	M	C(=)	
GS_Status			M

###### **13.2.4.11.3.1.2 Use of G\_Publish request**

The primitive G\_Publish request shall be used to initiate transfer of a publish data payload (GS\_Publish\_Data) to one or more subscribers. The publish data payload is stored in a local buffer and forwarded from the buffer to subscribers.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

The subscriber addressing shall be known through the lease identifier (GS\_Lease\_ID) that was obtained from the lease service.

Within the lease parameters, GS\_Resource shall be an identical value specified for a publisher and one or more subscribers in order to facilitate establishment of linkage between the endpoints.

GS\_Transfer\_Mode shall be provided with the request in order to indicate the quality of service and priority for the transfer of the data.

#### **13.2.4.11.3.1.3 Use of G\_Publish indication**

The primitive G\_Publish indication shall be used to signal the arrival of a publish data payload at a subscriber for processing.

The publish data payload (GS\_Publish\_Data) shall be delivered with the indication.

The subscriber session identifier (GS\_Session\_ID) and subscriber lease identifier (GS\_Lease\_ID) shall be returned to allow association of the indication primitive with a specific publish/subscribe relationship.

The indication shall be conditional on configuration-dependent timed delivery from the publisher and indicates fresh publication data.

#### **13.2.4.11.3.1.4 Use of G\_Publish confirm**

The primitive G\_Publish confirm shall be used to complete the G\_Publish request.

The publisher session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) shall be returned to allow matching of the confirm primitive with the original request primitive.

GS\_Status shall be returned to indicate success or failure of the operation, as described in Table 394.

**Table 394 – GS\_Status for G\_Publish confirm**

Value	Meaning
0	Success
1	Failure; lease has expired
2	Failure; other

#### **13.2.4.11.3.2 G\_Subscribe primitive**

##### **13.2.4.11.3.2.1 General**

Table 395 specifies parameter usage for the primitive G\_Subscribe.

**Table 395 – Primitive G\_Subscribe parameter usage**

Parameter name	G_Subscribe	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	
GS_Publish_Data		M
GS_Status		M

##### **13.2.4.11.3.2.2 Use of G\_Subscribe request**

The primitive G\_Subscribe request shall be used to retrieve the most recent publication data (GS\_Publish\_Data) from the local buffer.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

The publisher addressing shall be known through the lease identifier (GS\_Lease\_ID) that was obtained from the lease service.

#### **13.2.4.11.3.2.3 Use of G\_Subscribe confirm**

The primitive G\_Subscribe confirm shall be used to complete the G\_Subscribe request.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm primitive with the original request primitive.

GS\_Status shall be returned to indicate success or failure of the operation, as described in Table 396.

**Table 396 – GS\_Status for G\_Subscribe confirm**

Value	Meaning
0	Success; fresh data
1	Success; stale data
2	Failure; lease has expired
3	Failure; other

#### **13.2.4.11.3.3 G\_Publish\_Timer primitive**

##### **13.2.4.11.3.3.1 General**

Table 397 specifies parameter usage for the primitive G\_Publish\_Timer.

**Table 397 – Primitive G\_Publish\_Timer parameter usage**

Parameter name	G_PublishTimer
	Indication output
GS_Session_ID	M
GS_Lease_ID	M

#### **13.2.4.11.3.3.2 Use of G\_Publish\_Timer indication**

The primitive G\_Publish\_Timer indication shall be used within a publisher to signal the expiration of a publication timer.

The publisher session identifier (GS\_Session\_ID) and publisher lease identifier (GS\_Lease\_ID) shall be returned to allow association of the indication primitive with a specific publish/subscribe relationship.

#### **13.2.4.11.3.4 G\_Subscribe\_Timer primitive**

##### **13.2.4.11.3.4.1 General**

Table 398 specifies parameter usage for the primitive G\_Subscribe\_Timer.

**Table 398 – Primitive G\_Subscribe\_Timer parameter usage**

Parameter name	G_SubscribeTimer
	Indication output
GS_Session_ID	M
GS_Publish_Data	M
GS_Lease_ID	M

#### **13.2.4.11.3.4.2 Use of G\_Subscribe\_Timer indication**

The primitive G\_Subscribe\_Timer indication shall be used within a subscriber to signal the expiration of a subscription timer. The timer shall be reset by the G\_Publish indication.

The publish data payload (GS\_Publish\_Data) shall be delivered from the subscriber buffer with the indication.

The subscriber session identifier (GS\_Session\_ID) and subscriber lease identifier (GS\_Lease\_ID) shall be returned to allow association of the indication primitive with a specific publish/subscribe relationship.

#### **13.2.4.11.3.5 G\_Publish\_Watchdog primitive**

##### **13.2.4.11.3.5.1 General**

Table 399 specifies parameter usage for the primitive G\_Publish\_Watchdog.

**Table 399 – Primitive G\_Publish\_Watchdog parameter usage**

<b>Parameter name</b>	<b>G_Publish_Watchdog</b>
	<b>Indication output</b>
GS_Session_ID	M
GS_Publish_Data	M
GS_Lease_ID	M

#### **13.2.4.11.3.5.2 Use of G\_Publish\_Watchdog indication**

The primitive G\_Publish\_Watchdog indication shall be used within a subscriber to signal the expiration of a watchdog timer due to the absence of expected updates from a publisher. The timer is reset by the G\_Publish indication.

The now-stale publish data payload (GS\_Publish\_Data) shall be delivered from the subscriber buffer with the indication.

The session identifier (GS\_Session\_ID) and lease identifier (GS\_Lease\_ID) shall be returned to allow association of the indication primitive with a specific publish/subscribe relationship.

#### **13.2.4.12 Bulk transfer service**

##### **13.2.4.12.1 General**

The bulk transfer service provides for bulk data transfer. Bulk data transfer is used to transfer large items between gateway clients and wireless devices.

##### **13.2.4.12.2 Use of the service**

Bulk transfers operate in the context of a session between the GSAP service provider and the GSAP service user. All primitives supported by the gateway through a GSAP shall include the corresponding GS\_Session\_ID.

The client of the session shall manage the session-unique GS\_Transaction\_IDs for each primitive invoked by the client. This is necessary in order to maintain coordination between bulk transfer primitives.

The GS\_Lease\_ID, which represents the necessary communication resources allocated within the gateway, shall be supplied with each primitive.

Separate parallel bulk transfers shall be distinguished by a GS\_Transfer\_ID. A GS\_Transfer\_ID shall also be included in each GSAP service primitive. The transfer state is maintained for each bulk transfer in progress. For example, the block number being transferred is maintained by the endpoints.

G\_Bulk\_Open shall be used to open a bulk transfer. G\_Bulk\_Close shall be used to close a bulk transfer. G\_Bulk\_Transfer shall be used to perform the actual transfer of data segments within a bulk transfer.

### 13.2.4.12.3 Types of primitives and parameters

#### 13.2.4.12.3.1 G\_Bulk\_Open primitive

##### 13.2.4.12.3.1.1 General

Table 400 specifies parameter usage for the primitive G\_Bulk\_Open.

**Table 400 – Primitive G\_Bulk\_Open parameter usage**

<b>Parameter name</b>	<b>G_Bulk_Open</b>	
	<b>Request input</b>	<b>Confirm output</b>
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	
GS_Transfer_ID	M	
GS_Resource	M	
GS_Mode	M	
GS_Block_Size	M	M
GS_Item_Size	C	C
GS_Status		M

##### 13.2.4.12.3.1.2 Use of G\_Bulk\_Open request

The G\_Bulk\_Open request primitive shall be used to initiate a bulk transfer. The target device for a bulk transfer shall be implied by the GS\_Lease\_ID.

The target item for a bulk transfer shall be identified by GS\_Resource.

A transfer is directional (upload or download) and GS\_Mode defines the direction of the transfer. GS\_Mode = 0 specifies download and GS\_Mode = 1 specifies upload.

The GSAP service user sets GS\_Block\_Size to request a block size for the subsequent transfer phase.

The GSAP service user sets the GS\_Item\_Size to request download of an item of a particular size. The item may exceed the available download limits, resulting in an error response. GS\_Item\_Size = 0 requests the download of an item of indeterminate size.

##### 13.2.4.12.3.1.3 Use of G\_Bulk\_Open confirm

The G\_Bulk\_Open confirm primitive shall be used in response to the G\_Bulk\_Open request.

The GS\_Item\_Size shall be set by the GSAP service provider to indicate the item size. For a download, this shall be the maximum item size that will be accepted. For an upload, this shall be the actual item size. GS\_Item\_Size = 0 indicates that there is no limit imposed on the item size.

The GSAP service provider determines and returns the GS\_Block\_Size that will be used for the subsequent transfer phase. The block size may be reduced in size (based on available resources) from the original size requested in the G\_Bulk\_Open request.

GS\_Status shall indicate success or failure of the G\_Bulk\_Open, as shown in Table 401.

**Table 401 – GS\_Status for G\_Bulk\_Open confirm**

<b>Value</b>	<b>Meaning</b>
0	Success
1	Failure; item exceeds limits
2	Failure; unknown resource
3	Failure; invalid mode
4	Failure; other

### 13.2.4.12.3.2 G\_Bulk\_Transfer primitive

#### 13.2.4.12.3.2.1 General

Table 402 specifies parameter usage for the primitive G\_Bulk\_Transfer.

**Table 402 – Primitive G\_Bulk\_Transfer parameter usage**

Parameter name	G_Bulk_Transfer	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	
GS_Transfer_ID	M	
GS_Bulk_Data	C	C
GS_Status		M

#### 13.2.4.12.3.2.2 Use of G\_Bulk\_Transfer request

The G\_Bulk\_Transfer request primitive shall be used to move bulk data. GS\_Bulk\_Data shall be a transfer segment that is conditionally sent to the target in the case of a download.

G\_Bulk\_Transfer shall be used as many times as required to complete the transfer of a large item. G\_Bulk\_Close shall be used by the GSAP service user to indicate the completion of the transfer.

#### 13.2.4.12.3.2.3 Use of G\_Bulk\_Transfer confirm

The G\_Bulk\_Transfer confirm primitive shall be used in response to the G\_Bulk\_Transfer request. GS\_Bulk\_Data shall be a transfer segment that is conditionally received from the target in the case of an upload.

GS\_Status shall indicate success or failure of the G\_Bulk\_Transfer, as indicated in Table 403.

**Table 403 – GS\_Status for G\_Bulk\_Transfer confirm**

Value	Meaning
0	Success
1	Failure; communication failed
2	Failure; transfer aborted
3	Failure; other

### 13.2.4.12.3.3 G\_Bulk\_Close primitive

#### 13.2.4.12.3.3.1 General

Table 404 specifies parameter usage for the primitive G\_Bulk\_Close.

**Table 404 – Primitive G\_Bulk\_Close parameter usage**

Parameter name	G_Bulk_Close	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	
GS_Transfer_ID	M	
GS_Status		M

### **13.2.4.12.3.3.2 Use of G\_Bulk\_Close request**

The G\_Bulk\_Close request primitive shall be used to complete a bulk transfer, and to clean up any resources or state handling necessary in the GSAP service provider.

### **13.2.4.12.3.3.3 Use of G\_Bulk\_Close confirm**

The G\_Bulk\_Close confirm primitive shall be used in response to the G\_Bulk\_Close request. GS\_Status indicates success of the G\_Bulk\_Close, as shown in Table 405.

**Table 405 – GS\_Status for G\_Bulk\_Close confirm**

Value	Meaning
0	Success

## **13.2.4.13 Alert service**

### **13.2.4.13.1 General**

The alert service provides for the establishment of alert notification events for gateway clients. Additional operations may be required to collect additional information related to the alert or to respond to the alert.

### **13.2.4.13.2 Use of the service**

Alert services operate in the context of a session between the GSAP service provider and the GSAP service user. All primitives supported by the gateway through a GSAP shall include the corresponding GS\_Session\_ID.

The client of the session shall manage the session-unique outstanding GS\_Transaction\_IDs for each primitive it invokes. This is necessary in order to maintain coordination between alert primitives.

The GS\_Lease\_ID, which represents the necessary gateway entity and communication resources, shall be supplied with each primitive.

G\_Alert\_Subscription shall be used to subscribe to alerts either by category or by specific alerts.

### **13.2.4.13.3 Types of primitives and parameters**

#### **13.2.4.13.3.1 G\_Alert\_Subscription primitive**

##### **13.2.4.13.3.1.1 General**

Table 406 specifies parameter usage for the primitive G\_Alert\_Subscription.

**Table 406 – Primitive G\_Alert\_Subscription parameter usage**

<b>Parameter name</b>	<b>G_Alert_Subscription</b>	
	<b>Request input</b>	<b>Confirm output</b>
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Lease_ID	M	
GS_Subscription_List	M	C
GS_Category	C	C
GS_Network_Address	C	C
GS_Alert_Source_ID	C	C
GS_Subscribe	M	C
GS_Enable	M	C
GS_Status		M

### **13.2.4.13.3.1.2 Use of G\_Alert\_Subscription request**

The G\_Alert\_Subscription request primitive shall be used to manage an alert subscription list.

GS\_Subscription\_List shall contain one or more alert subscription modification requests. Each list element can be used to modify the subscription for a particular category of alerts or to modify the subscription for a specific alert from a specific source.

List elements may specify the GS\_Category to indicate subscription modification for a particular category of alerts. Alert categories include:

- 0 = device;
- 1 = network;
- 2 = security; and
- 3 = process.

GS\_Network\_Address and GS\_Alert\_Source\_ID shall not be supplied if GS\_Category is supplied.

Alternatively, list elements may specify GS\_Network\_Address and GS\_Alert\_Source\_ID instead of GS\_Category to specify a specific device and an identifier of a specific alert from that device.

NOTE It is anticipated that some gateway clients, such as full alarm management systems, will use complete categories, whereas other gateway clients may only require a select subset of alerts.

GS\_Subscribe and GS\_Enable control the actions for each list element. GS\_Subscribe shall be used to specify which alerts are to be received within the gateway entity and forwarded to the GSAP service user in the form of G\_Alert\_Notification indications. GS\_Enable shall be used to control the underlying generation of alerts at the source. GS\_Subscribe = 1 subscribes to a specific alert or an alert category, while GS\_Subscribe = 0 unsubscribes from the alert. GS\_Enable = 1 enables a specific alert or an alert category, while GS\_Enable = 0 disables the alert at the source.

In order to synchronize alarm state between alarm source and the gateway entity, alarm recovery shall be initiated on subscriptions.

### **13.2.4.13.3.1.3 Use of G\_Alert\_Subscription confirm**

The G\_Alert\_Subscription confirm primitive shall be used in response to the G\_Alert\_Subscription request.

GS\_Status shall indicate success or failure of the G\_Alert\_Subscription request, as indicated in Table 407. A GS\_Subscription\_List with a single element shall be conditionally returned if the operation fails. The status code relates to the particular element. Processing of the list shall stop at the first failed element.

**Table 407 – GS\_Status for G\_Alert\_Subscription confirm**

Value	Meaning
0	Success
1	Failure; Invalid category
2	Failure; Invalid individual alert
3	Failure; other

### **13.2.4.13.3.2 G\_Alert\_Notification primitive**

#### **13.2.4.13.3.2.1 General**

Table 408 specifies parameter usage for the primitive G\_Alert\_Notification.

**Table 408 – Primitive G\_Alert\_Notification parameter usage**

<b>Parameter name</b>	<b>G_Alert_Notification</b>
	<b>Indication output</b>
GS_Session_ID	M
GS_Lease_ID	M
GS_Alert	M
GS_Network_Address	M
GS_Alert_Source_ID	M
GS_Time	M
GS_Class	M
GS_Direction	M
GS_Category	M
GS_Type	M
GS_Priority	M
GS_Alert_Data	C

**13.2.4.13.3.2.2 Use of G\_Alert\_Notification indication**

The G\_Alert\_Notification indication shall be generated by the GSAP service provider and sent to the GSAP user in response to an alert received by the gateway. Notification shall only be provided for those alerts to which the GSAP client had subscribed, and for which notification has been enabled.

A GS\_Alert structure shall be provided within the indication to provide alert specific details as follows:

- GS\_Network\_Address indicates the source device of the alert.
- GS\_Alert\_Source\_ID indicates the specific alert within the source device.
- GS\_Time is a timestamp that indicates when the alert was originally generated.
- GS\_Class = 0 identifies the alert as an event; GS\_Class = 1 identifies the alert as an alarm.
- GS\_Direction further classifies alarms as follows:
  - 0 = alarm condition ended.
  - 1 = alarm condition began.
- GS\_Category defines the alert category as follows:
  - 0 = process-related.
  - 1 = device-related.
  - 2 = network-related.
  - 3 = security-related.
- GS\_Type defines sub-categories for alerts. The actual value is application-specific.
- GS\_Priority defines a priority for the alert. Larger values indicate higher priority. The actual value is application-specific.
- GS\_Alert\_Data allows inclusion of alert-related information. This field is conditional on whether additional alert information is available. The actual value is application-specific.

The gateway entity shall acknowledge alert receipt.

**13.2.4.14 Gateway configuration service****13.2.4.14.1 General**

The gateway configuration service provides for reading and writing the gateway configuration attributes.

### 13.2.4.14.2 Use of the service

The gateway client uses the G\_Read\_Gateway\_Configuration primitive to retrieve gateway configuration attributes.

### 13.2.4.14.3 Types of primitives and parameters

#### 13.2.4.14.3.1 G\_Read\_Gateway\_Configuration primitive

##### 13.2.4.14.3.1.1 General

Table 409 specifies parameter usage for the primitive G\_Read\_Gateway\_Configuration.

**Table 409 – Primitive G\_Read\_Gateway\_Configuration parameter usage**

Parameter name	G_ReadGatewayConfiguration	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Attribute_Identifier	M	
GS_Attribute_Value		C
GS_Status		M

##### 13.2.4.14.3.1.2 Use of G\_Read\_Gateway\_Configuration request

The gateway client uses the primitive G\_Read\_Gateway\_Configuration request to retrieve gateway configuration parameters.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

The requested attribute is specified by the attribute identifier (GS\_Attribute\_Identifier), as shown in Table 410. The requested value is specified by the attribute value (GS\_Attribute\_Value).

**Table 410 – GS\_Attribute\_Identifier values**

Value	Meaning
0	GS_GUID
1	GS_Max_Retries
2	GS_Max_Devices
3	GS_Actual_Devices

##### 13.2.4.14.3.1.3 Use of G\_Read\_Gateway\_Configuration confirm

The gateway entity uses the primitive G\_Read\_Gateway\_Configuration confirm to complete the G\_Read\_Gateway\_Configuration request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm with the original request.

If the operation succeeds, the value (GS\_Attribute\_Value) is returned for the requested attribute (GS\_Attribute\_Identifier).

GS\_Status is returned to indicate success or failure of the operation, as described in Table 411.

**Table 411 – GS\_Status for G\_Read\_Gateway\_Configuration confirm**

Value	Meaning
0	Success
1	Failure

### 13.2.4.14.3.2 G\_Write\_Gateway\_Configuration primitive

#### 13.2.4.14.3.2.1 General

Table 412 specifies parameter usage for the primitive G\_Write\_Gateway\_Configuration.

**Table 412 – Primitive G\_Write\_Gateway\_Configuration parameter usage**

Parameter name	G_Write_Gateway_Configuration	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Attribute_Identifier	M	
GS_Attribute_Value	M	
GS_Status		M

#### 13.2.4.14.3.2.2 Use of G\_Write\_Gateway\_Configuration request

The gateway client uses the primitive G\_Write\_Gateway\_Configuration request to alter gateway configuration attributes.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

The requested attribute is specified by the attribute identifier (GS\_Attribute\_Identifier), as shown in Table 413. The requested value is specified by the attribute value (GS\_Attribute\_Value).

**Table 413 – GS\_Attribute\_Identifier values**

Value	Meaning
0	GS_GUID
1	GS_Max_Retries

#### 13.2.4.14.3.2.3 Use of G\_Write\_Gateway\_Configuration confirm

The gateway entity uses the primitive G\_Write\_Gateway\_Configuration confirm to complete the G\_Write\_Gateway\_Configuration request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm primitive with the original request primitive.

GS\_Status is returned to indicate success or failure of the operation, as described in Table 414.

**Table 414 – GS\_Status for G\_Write\_Gateway\_Configuration confirm**

Value	Meaning
0	Success
1	Failure; invalid attribute value
2	Failure; other

### 13.2.4.15 Device configuration service

#### 13.2.4.15.1 General

The device configuration service provides a method to manage the configuration of the devices that are associated with a gateway. This is useful for commissioning wireless devices for host systems and related applications.

The device configuration has services to write and to read back the configuration for one or more devices.

A unique identifier is used to match the configuration to a specific device. A network address is specified for usage in configuration of the device, allowing subsequent logical access of the device.

The device list report service shall be used to determine the devices associated with the gateway. This service works in conjunction with the device list report service by providing the ability to limit the devices that are associated with a gateway.

An optional configuration file is provided for each device. The format of the configuration file is network implementation dependent. Information contained in this file is intended to allow gateways to automatically provision devices to join the network. Further configuration is accomplished one-on-one by client server and bulk transfer services.

If the gateway is used to provision devices, the device list report will be empty until devices are provisioned and join the system.

#### 13.2.4.15.2 Use of the service

The gateway client uses the G\_Write\_Device\_Configuration primitive to set the configuration for devices associated with a gateway entity.

#### 13.2.4.15.3 Types of primitives and parameters

##### 13.2.4.15.3.1 G\_Write\_Device\_Configuration primitive

###### 13.2.4.15.3.1.1 General

Table 415 specifies parameter usage for the primitive G\_Write\_Device\_Configuration.

**Table 415 – Primitive G\_Write\_Device\_Configuration parameter usage**

Parameter name	G_Write_Device_Configuration	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List	M	
GS_Configure	M	
GS_Undevice_ID	M	
GS_Network_Address	M	
GS_Provisioning_Info	U	
GS_Status		M

###### 13.2.4.15.3.1.2 Use of G\_Write\_Device\_Configuration request

The gateway client uses the primitive G\_Write\_Device\_Configuration request to configure the devices associated with a gateway entity.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

A list of devices associated with the gateway entity (GS\_Device\_List) is supplied. For each device in the list, the unique device identifier (GS\_Undevice\_ID) indicates the device associated with the configuration. If GS\_Configure = 1, the configuration is added for the specific device, while if GS\_Configure = 0, the configuration is removed for the specific device. A matching network address (GS\_Network\_Address) indicates the logical address to associate with the device.

Optional device provisioning information (GS\_Provisioning\_Info) shall be supplied to the gateway for the gateway to control provisioning of the device.

#### **13.2.4.15.3.1.3 Use of G\_Write\_Device\_Configuration confirm**

The gateway entity uses the primitive G\_Write\_Device\_Configuration confirm to complete the G\_Write\_Device\_Configuration request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm with the original request.

GS\_Status is returned to indicate success or failure of the operation, as described in Table 416.

**Table 416 – GS\_Status for G\_Write\_Device\_Configuration confirm**

Value	Meaning
0	Success
1	Failure; invalid or duplicate network address
2	Failure; out of memory
3	Failure; maximum gateway devices exceeded
4	Failure; provisioning information invalid
5	Failure; other

#### **13.2.4.15.3.2 G\_Read\_Device\_Configuration primitive**

##### **13.2.4.15.3.2.1 General**

Table 417 specifies parameter usage for the primitive G\_Read\_Device\_Configuration.

**Table 417 – Primitive G\_Read\_Device\_Configuration parameter usage**

Parameter name	G_Read_Device_Configuration	
	Request input	Confirm output
GS_Session_ID	M	M(=)
GS_Transaction_ID	M	M(=)
GS_Device_List	U	M
GS_Undevice_ID	U	M
GS_Network_Address		M
GS_Provisioning_Info		U
GS_Status		M

##### **13.2.4.15.3.2.2 Use of G\_Read\_Device\_Configuration request**

The gateway client uses the primitive G\_Read\_Device\_Configuration request to retrieve the configuration of the devices associated with a gateway entity.

A session identifier (GS\_Session\_ID) shall be obtained from the G\_Session service and included in the request.

A session unique transaction identifier (GS\_Transaction\_ID) shall be specified for each invocation of the service.

A list of devices associated with the gateway entity (GS\_Device\_List) is optionally supplied. If no list is supplied, the request is for all devices. If a list is supplied, the request is for those specific devices. If a list is supplied, the unique device identifier (GS\_Undevice\_Device\_ID) indicates the device associated with the configurations to be read.

#### **13.2.4.15.3.2.3 Use of G\_Read\_Device\_Configuration confirm**

The gateway entity uses the primitive G\_Read\_Device\_Configuration confirm to complete the G\_Read\_Device\_Configuration request to the gateway client.

The session identifier (GS\_Session\_ID) and transaction identifier (GS\_Transaction\_ID) are returned to allow matching of the confirm with the original request.

A list of devices associated with the gateway entity (GS\_Device\_List) is returned. For each device in the list, the unique device identifier (GS\_Undevice\_Device\_ID) indicates the device associated with the configuration. A matching network address (GS\_Network\_Address) indicates the logical address associated with the device. The optional device configuration file (GS\_Provisioning\_Info) indicates provisioning information for the device.

GS\_Status is returned to indicate success or failure of the operation, as described in Table 418.

**Table 418 – GS\_Status for G\_Read\_Device\_Configuration confirm**

Value	Meaning
0	Success
1	Failure; other

### **13.3 Protocol**

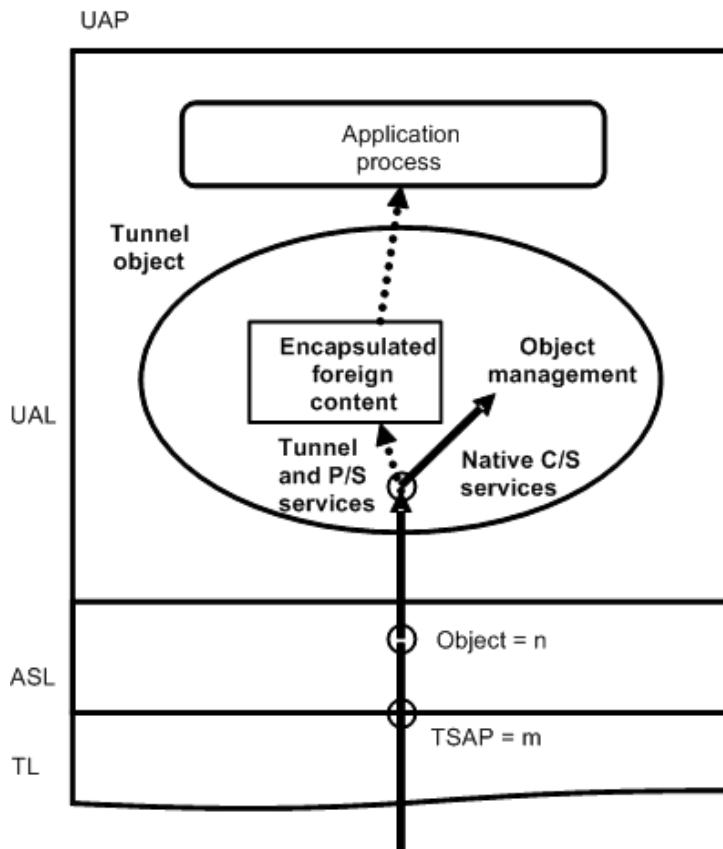
#### **13.3.1 Tunneling**

##### **13.3.1.1 General**

The tunnel object (TUN) is a native object that acts as a communication endpoint for the following messaging:

- Encapsulated foreign protocol content (shown in Figure 150 as a dotted line); and
- Native service content (shown in Figure 150 as a solid line) to configure and manage the tunnel object.

Gateway processes and adapter processes use tunnel objects to support foreign protocol translation. An important aspect of the TUN object is that it provides buffered message behavior for foreign content.



**Figure 150 – Tunnel object model**

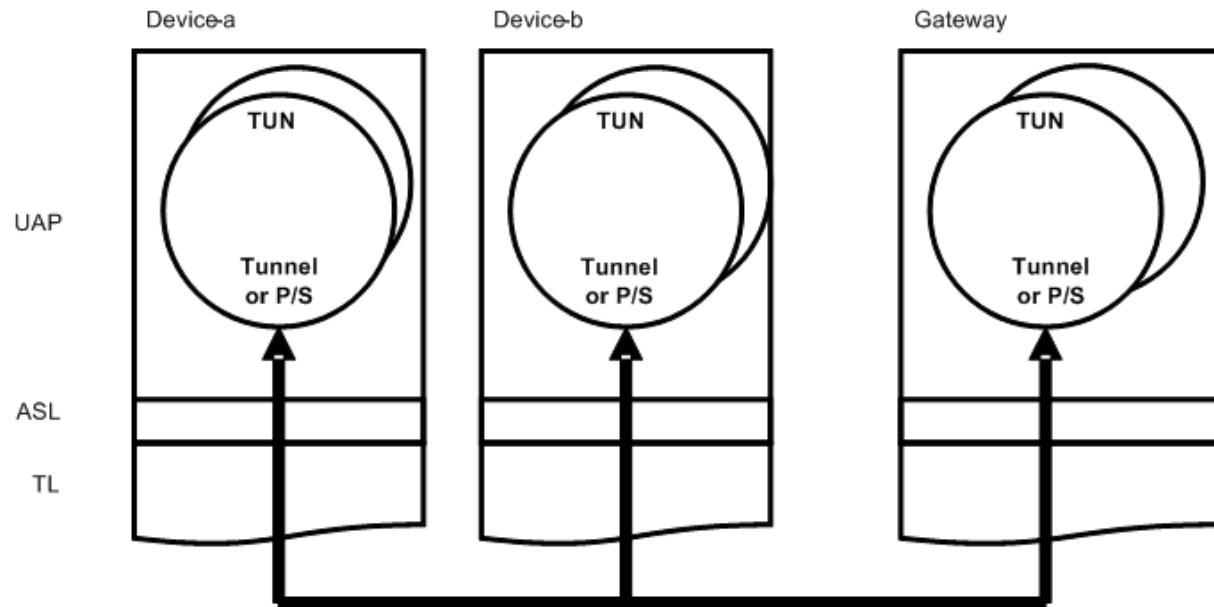
One or more TUNs may exist within a user application process (UAP).

Each TUN object can handle a complete foreign protocol or a portion thereof. Devices that handle multiple foreign protocols will need to implement multiple TUNs. The TUN object is independent of the foreign protocol.

The TUN object relies on the application sub-layer (ASL) in order to route messages between peer TUNs and between a TUN object and other non-TUN objects.

### 13.3.1.2 Distributing tunnel objects

Each device may have one or more TUNs. Tunneling devices shall include at least one TUN object as an endpoint for tunnels. Figure 151 shows a group of related devices with tunnel endpoints interconnected between TUNs.



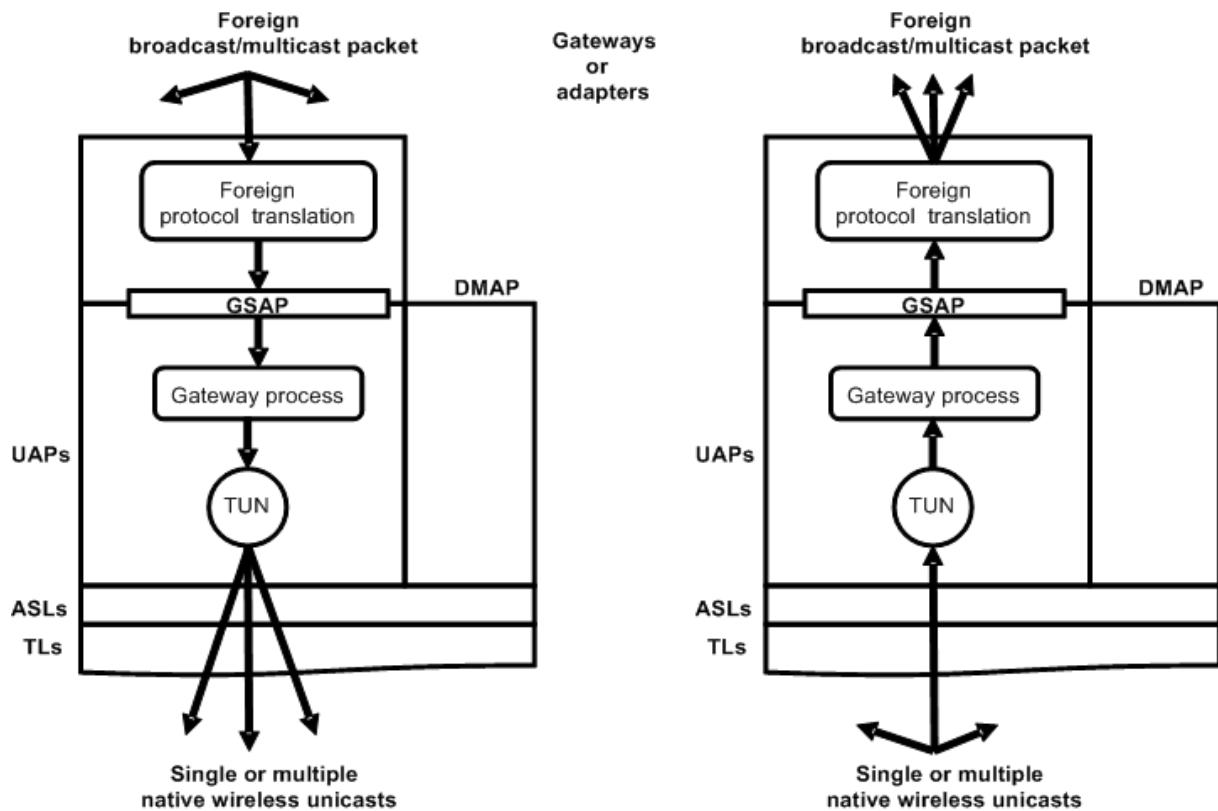
**Figure 151 – Distributed tunnel endpoints**

Field devices and adapters may contain TUNs that cooperate with other TUNs in a gateway. A group of related TUNs communicate via a common foreign protocol. The typical usage of TUNs is to communicate between end devices and a host system via the gateway. Direct device-to-device tunneling is also supported within the object model.

TUN object communication is established by utilizing transport layer (TL) services invoked and augmented via the ASL. Communication relationships include publish, subscribe, 2-part tunnel, and 4-part tunnel. Multiple relationships may be established simultaneously.

#### 13.3.1.3 Multicast, broadcast, and one-to-many messaging

As shown in Figure 152, foreign protocols may require translation of broadcast/multicast messaging relationships when utilizing services such as publish/subscribe and alert distribution. This messaging requires translation support within this standard.



**Figure 152 – Multicast, broadcast, and one-to-many messaging**

This standard provides one-to-many messaging support in the tunnel object in order to support translation of the foreign protocol multicast and broadcast requests. The underlying layers of the protocol suite do not provide broadcast or multicast services to the application layer. One-to-many messaging shall be achieved via a series of unicast operations. Protocol translation applications shall not rely on simultaneous delivery of the unicast messages.

#### 13.3.1.4 Tunnel buffered message behavior

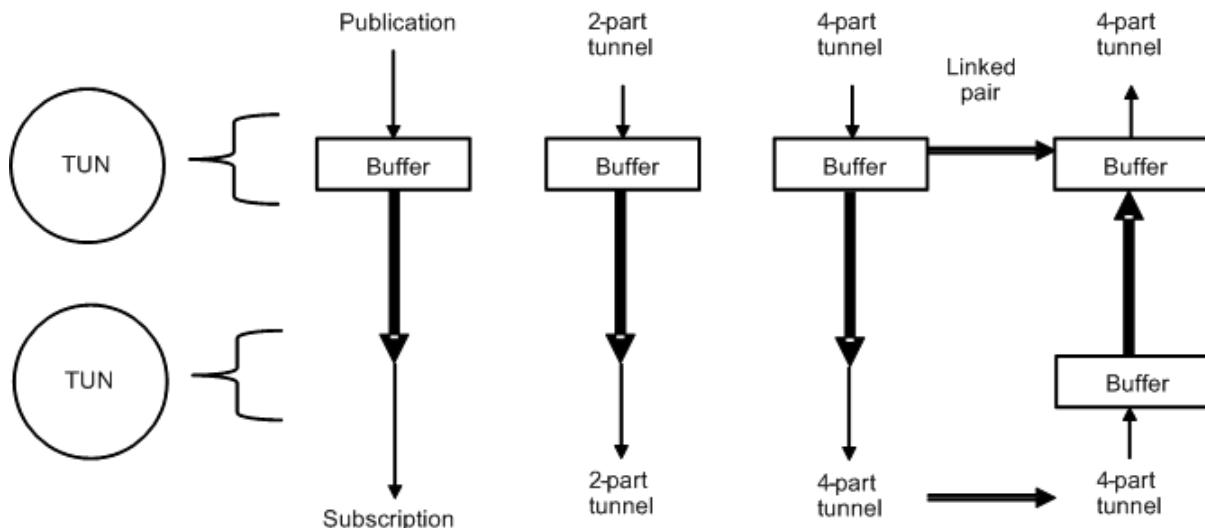
TUN object communication shall provide capabilities for buffered and non-buffered behavior for publish/subscribe and tunnel-based message exchanges. TUNs shall be capable of cooperatively managing buffered behavior to reduce wireless transactions.

NOTE 1 Legacy protocols may utilize buffered messaging exchanges to support energy efficient and high performance protocol translation.

NOTE 2 Certain applications may not tolerate buffered behavior due to safety and synchronization requirements.

NOTE 3 Buffers are a single element deep. Nothing in this standard prevents implementation of caching and queuing enhancements.

As shown in Figure 153, each endpoint of a communication flow has a different buffering responsibility, depending on the relationship.



**Figure 153 – Tunnel object buffering**

In Figure 153, two TUN objects are shown. The thin arrows indicate interactions from tunnel applications that utilize the objects. The thick arrows indicate message flows across the network between the objects. Three types of transaction are shown, a publish/subscribe transaction, a 2-part tunnel transaction, and a 4-part tunnel transaction. Buffers are shown to illustrate the buffered messaging behavior between tunnel endpoints.

A publisher and a subscriber are linked from TUN object to TUN object for periodic updates. Publishers use change of state buffer publications to avoid sending repeated information; subscribers tolerate limited intervals with missing publications.

Messaging with the 2-part tunnel service behaves in a similar manner, except that the messaging is aperiodic.

Messaging with the 4-part tunnel service distinguishes a request and a response side via the request/response bit in the tunnel service header. The request side buffers the first request and also forwards the request. A response is generated as indicated by the double arrow on the response side in Figure 153. The response is stored in a second linked buffer on the request side, as indicated by the double arrow in Figure 153. Change of state processing applies to subsequent duplicate requests, wherein the response is returned from the local buffer. Where change of state indicates an altered request, the request is forwarded and the local response buffer is updated. A final buffer is shown in Figure 153 on the response side. This buffer supports change of state behavior wherein a single request results in multiple responses over time to update the request side.

### 13.3.1.5 Tunnel object attributes

TUN object attributes are described in Clause 12, but are further described herein.

The Protocol attribute is used to configure the protocol associated with the tunnel object and the associated remote tunnel objects. When the protocol is set to none, the tunnel can be configured. Once another protocol is set, the tunnel object configuration is applied and the status is updated to reflect the result.

The tunnel endpoint structure specifies address information pointing to a single remote tunnel object. The array of tunnel endpoints allows specification of one or more tunnel endpoints representing remote tunnel objects. This allows a single communication relationship to span multiple tunnel objects where necessary. Max\_Peer\_tunnels indicates the maximum number of entries in the array. Num\_Peer\_Tunnels indicates the actual number of entries configured in the array.

One of several types of communication flow types is selected between the tunnel objects by configuration of the Flow\_Type attribute. Flow types include 2-part tunnel, 4-part tunnel, publish, and subscribe.

For publish and subscribe Flow\_Types, the Update\_Policy allows configuration of periodic publication or change of state publication. Periodic publication occurs on every opportunity. CoSt publication occurs only when fresh publication data is available. The publications frequency is based on the Period attribute. The actual timing is based on a combination of the Period and the Phase attributes. The Stale\_Limit is used in the subscriber to configure behavior for detection of excessive publication loss or delay. Stale\_Limit is a multiplier that configures the number of periods that a subscriber will wait before considering lost publications to indicate a problem.

Foreign\_Destination\_Address and Foreign\_Source\_Address are the addresses associated with the tunnel endpoint by the foreign protocol. The format is dependent on the foreign protocol. These addresses are returned to protocol translator applications as tunnel object messages are received. They allow utilization of network addressing as defined in this standard in lieu of carrying the foreign addressing. Mapping via the tunnel object allows reconstruction of foreign PDUs containing address information.

**NOTE** Depending on the specific foreign protocol conversion, the foreign PDU will vary. Most fieldbus protocols will form DPDUs for direct delivery on a local link. IP-based protocols may form NPDUs, where a final encapsulation is achieved by an address resolution protocol.

Connection\_Info[] and Transaction\_Info[] are octet strings that are written by the protocol translator as required. Connection\_Info[] is utilized to provide protocol specific static message content on message receipt in order to eliminate the repeated wireless message transfer of the content. Transaction\_Info[] is utilized to provide protocol specific message content on receipt of a response, where the content would otherwise be echoed from the request in the response, eliminating the wireless transfer of the content. Further description is provided in 13.3.1.9 and Annex O.

It is the responsibility of the TUN object implementation to maintain a related contract for each tunnel endpoint.

### **13.3.1.6 Tunnel object messaging**

#### **13.3.1.6.1 Application sub-layer service usage**

TUN objects provide connection services that include a publish/subscribe service, a 2-part tunnel service, and a 4-part tunnel service. Each service has a buffered and a non-buffered mode of operation.

The optional external interface for invoking the gateway connection services is described in 13.2.

The TUN object utilizes the ASL to deliver and receive service content as described for the publish service in 12.17.3.2 and the tunnel service in 12.17.6.2. The ASL provides object-to-object delivery of publish/subscribe payloads in external formats through the publish request primitive. The ASL also provides a linked tunnel request and tunnel response primitive.

The header is described in 12.22.2.3. This header enables request and response specification, service type specification (publish or tunnel), and object identifier addressing mode (4-bit, 8-bit, or 16 bit). A large number of tunnel objects will result in a larger address space and more overhead in the header.

The publish service payload format is described in 12.22.2.12 by Table 348. There is no explicit length in the header. The length of the publication shall be supplied with the publish request and shall be known to the subscriber by information supplied with the indication.

The tunnel service request and response payload formats are described in 12.22.2.9. The request allows 7-bit length (0-127 byte payloads) or 15-bit length (128-32 767 byte payloads). Inclusion of the length allows tunnel message to be concatenated by the ASL.

**NOTE** Most encapsulated messages from legacy protocols referenced by this standard fall into the range of less than a 127-byte payload, resulting in a 7-bit field.

#### **13.3.1.6.2 Information classification and transfer rules**

From a caching and buffering viewpoint, information may be classified as constant, static, dynamic, or non-bufferable. These classifications are described in 12.6.3 for native object

attributes. The same guidance applies to the selection of buffering for publish and tunnel services for foreign payloads.

Constant information should not be transferred more than once between TUNs, except where local copies are lost due to power cycling, reset, cache deletion, or elimination of references to the information.

Static information should not be transferred more than once between TUNs, except as indicated for constant information and where the static information has been modified.

Dynamic information should only be transferred between TUNs when its value has changed unless it is required more often to indicate that the source or destination is still active.

Non-bufferable information should be transferred between TUNs on each request.

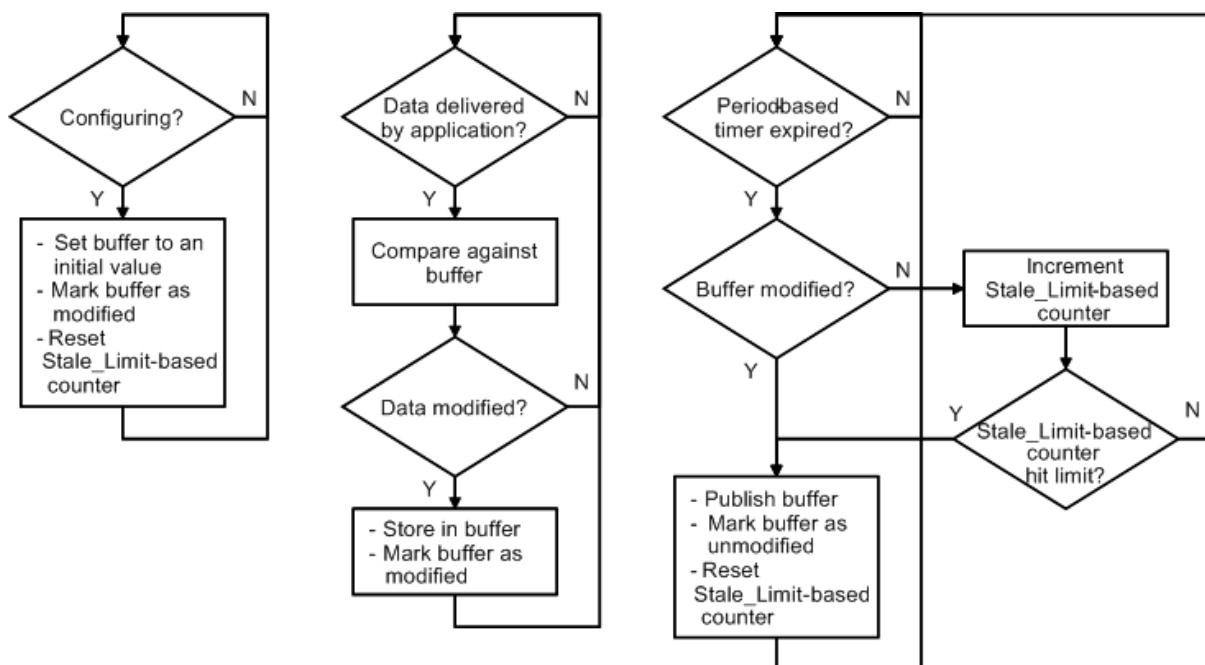
#### 13.3.1.6.3 Publish/subscribe service

The tunnel object provides facilities to accomplish buffered and non-buffered publish/subscribe messaging for dynamic information update.

The flowcharts below describe the behavior of TUN object publishers and subscribers that utilizes buffering. The behavior describes the base message transfer agreement between a publisher and a subscriber based on the TUN object attribute configurations.

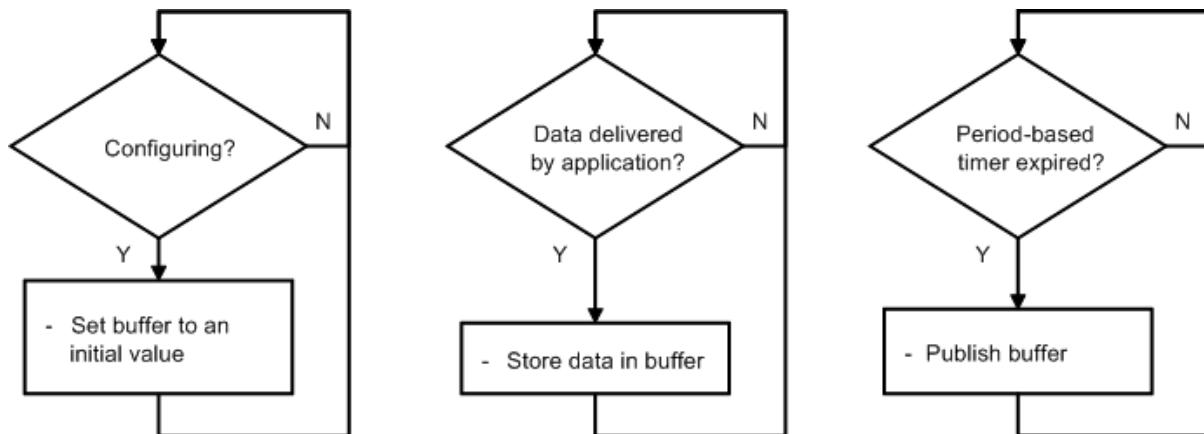
**NOTE** The interpretation and actions for initial, stale, and repeat data is based on implementation, as is the CoSt algorithm.

The publish/subscribe publisher connection shall operate as shown in Figure 154 when CoSt updates are configured.

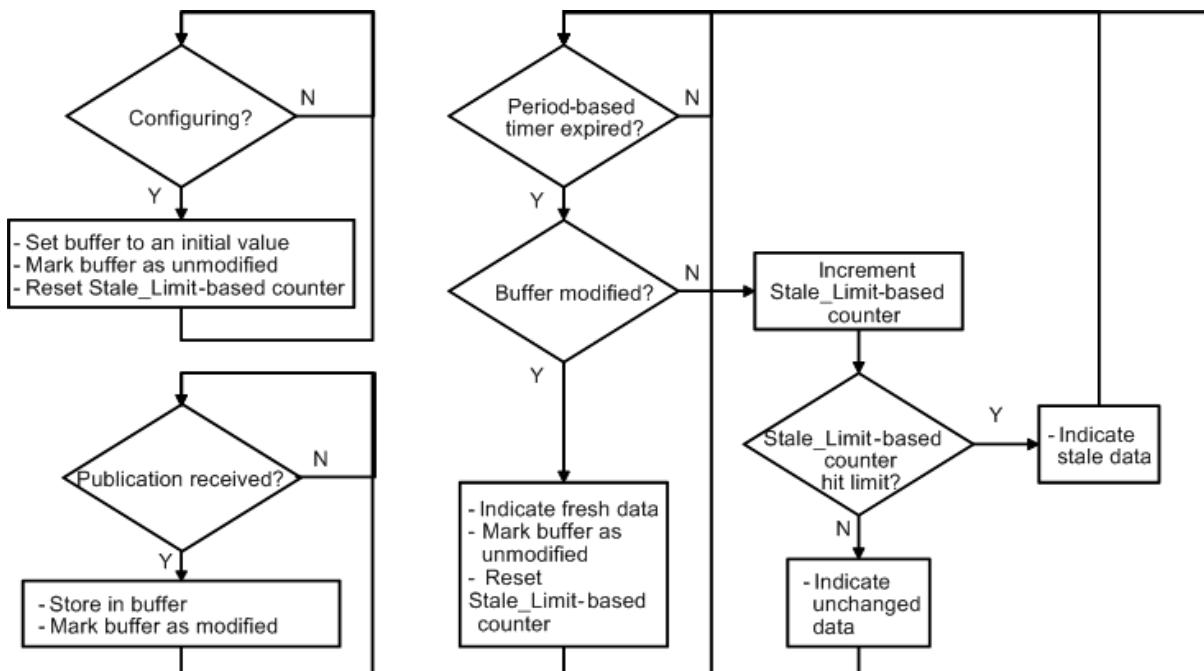


**Figure 154 – Publish/subscribe publisher CoSt flowchart**

The publish/subscribe publisher connection operates as shown in Figure 155 when periodic updates are configured.

**Figure 155 – Publish/subscribe publisher periodic flowchart**

The publish/subscribe subscriber connection operates as shown in Figure 156 when periodic or CoSt updates are configured.

**Figure 156 – Publish/subscribe subscriber common periodic and CoSt flowchart**

#### 13.3.1.6.4 Tunnel service

The tunnel object provides facilities to accomplish buffered and non-buffered tunnel service messaging. Non-buffered tunnel service messaging provides support for unconditional transfer of non-bufferable and constant information. Buffered tunnel service messaging provides support for buffering and contingent transfer of static and dynamic information.

#### 13.3.1.7 Multiple server responses

Certain client/server requests receive multiple responses. One reason is that the request requires extended processing and an immediate response is sent which indicates that the request was received and that the real response will be sent after processing is complete. This is known in some protocols as a delayed response. In other cases, the server provides additional updates over time to satisfy the initial request. Certain protocols collect process variables or historian information in this manner.

The client/server buffered and unbuffered services support multiple application responses for these purposes. In the case of the buffered response, the read buffer maintains the latest response. The client receives an indication on each response.

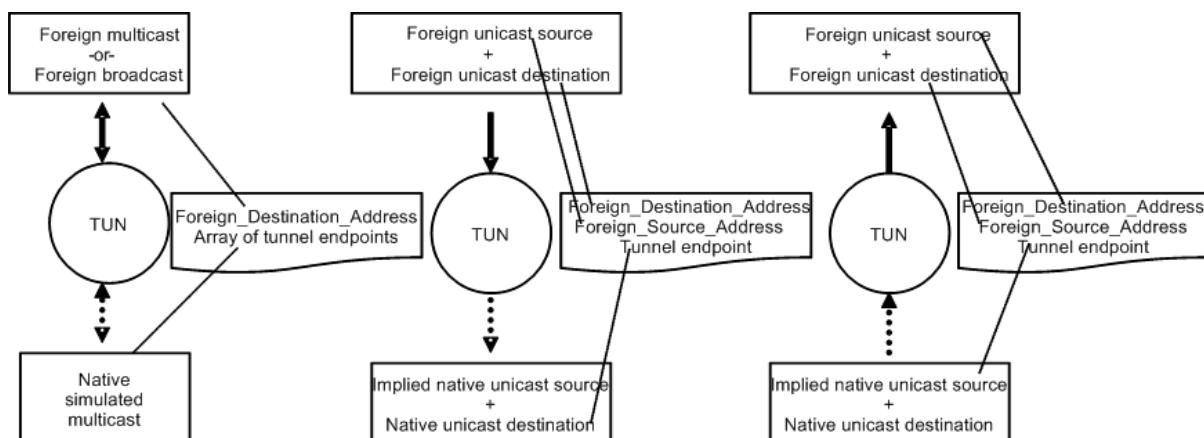
### 13.3.1.8 Tunnel object address mapping

The TUN object contains three address fields (Foreign\_Destination\_Address, Foreign\_Source\_Address, and the Array of Tunnel endpoints) that are utilized in the translation between foreign addresses and native addresses.

As shown in Figure 157, foreign multicast and foreign broadcast addresses require translation to native addresses and messaging.

The first case is where the multicast or broadcast originates on the foreign network. Since multiple hosts, protocols, or applications may share a wireless network as described herein, sending a foreign broadcast to all wireless devices is inefficient. Thus, foreign broadcast into the wireless network shall utilize simulated multicast to a limited group. The TUN object shall be utilized to simulate multicast delivery (one-to-many messaging) by maintaining a list of unicast addresses (array of tunnel endpoints) and by utilizing a sequence of unicast deliveries.

The second case is where the multicast or broadcast originates on the wireless network and is destined for the foreign network. A single APDU is delivered from the wireless network and acted on by a protocol translator to generate a multicast or broadcast PDU on the foreign network.



**Figure 157 – Network address mappings**

Also shown in Figure 157 is a pair of unicast address translations.

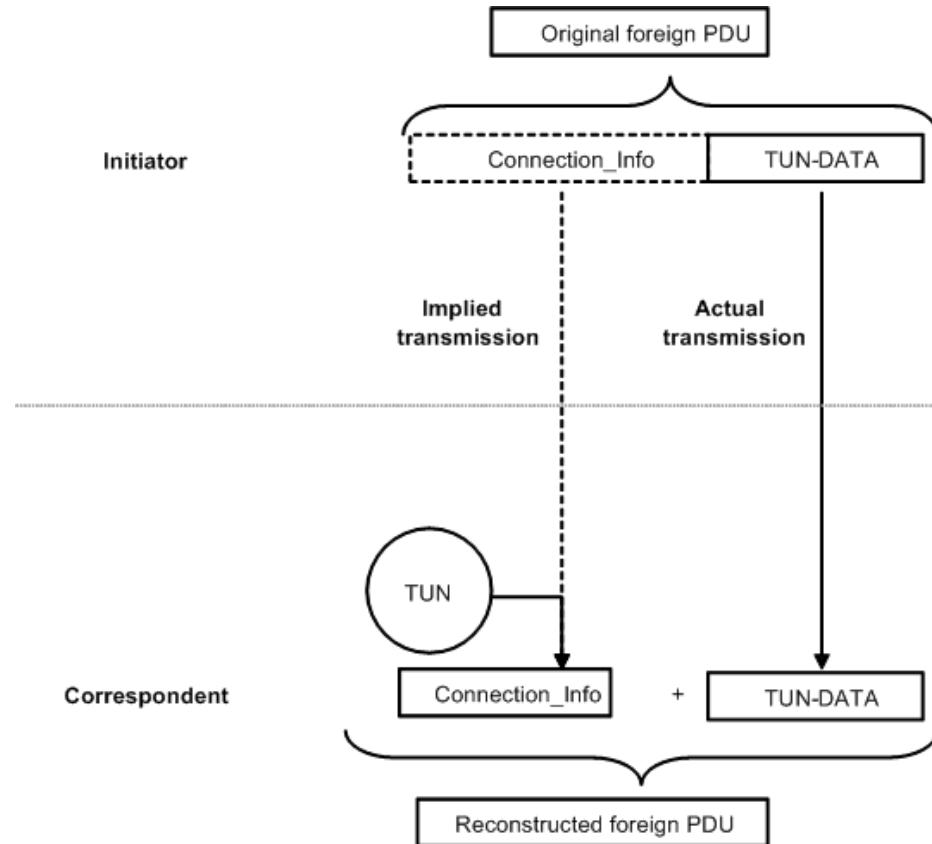
The first case translates from a foreign source/destination address pair to a native address pair. The second case translates from a native source/destination address pair to a foreign address pair. Both the Foreign\_Destination\_Address and the Foreign\_Source\_Address are utilized. Only the address information from a single tunnel endpoint is necessary, since the TUN object has access to its own native address for usage in source or destination fields. Foreign source and destination definition depends on the direction of the transfer.

### 13.3.1.9 Connection and transaction information

TUN objects function as initiator endpoints (publisher and tunnel request) and correspondent endpoints (subscriber and tunnel response). Protocol translation sends foreign content as TUN-DATA between the endpoints. Since most legacy protocols are not optimized for low-energy wireless communication, optional mechanisms are available to increase efficiency.

When a protocol translator tunnels a foreign PDU, it is not efficient to send static portions of the foreign PDU between the endpoints. Such static information includes preambles and secondary fixed addressing, such as logical unit identifiers. As shown in Figure 158, TUN objects provide an optional generic mechanism (Connection\_Info) for provision of static information on foreign PDU receipt without wireless transfer of the static information.

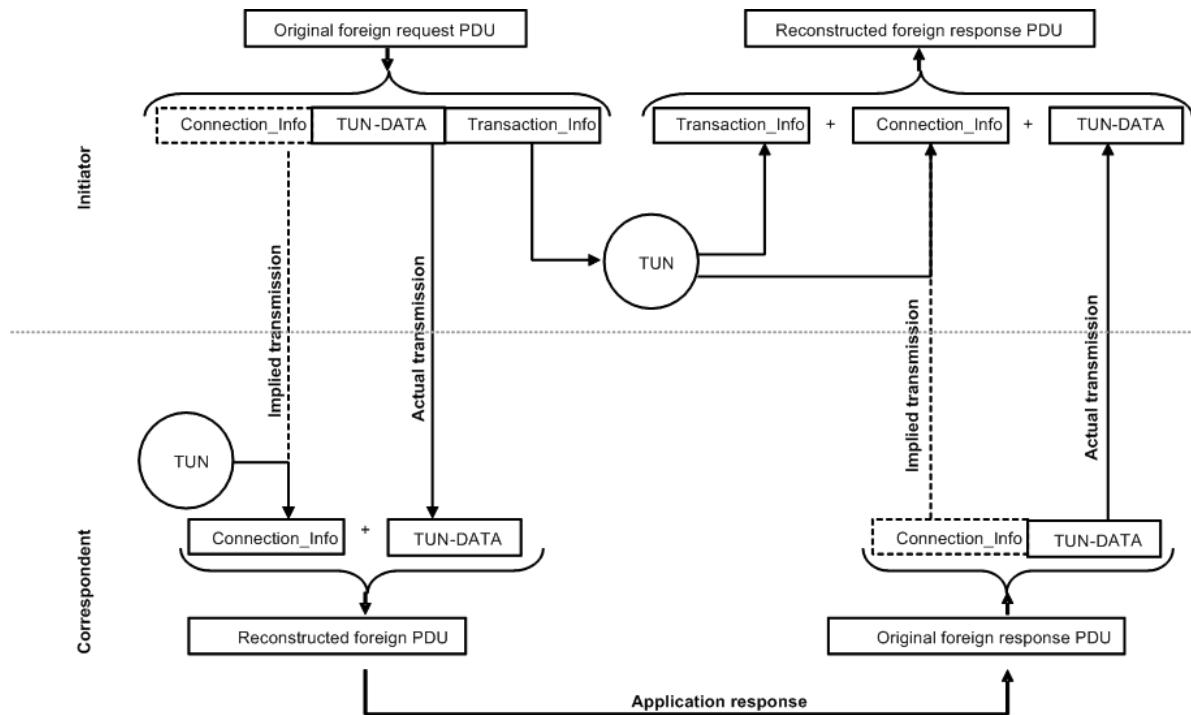
NOTE Depending on the specific foreign protocol conversion, the foreign PDU will vary. Most fieldbus protocols will form DPDUs for direct delivery on a local link. IP-based protocols may form NPDUs, where a final encapsulation is achieved by an address resolution protocol.



**Figure 158 – Connection\_Info usage in protocol translation**

When a protocol translator performs a transaction, it is not efficient to carry transaction-specific information that is only used to identify the transaction at the initiator. Such information includes information to link the original request to the response, where knowledge of the endpoint can be utilized. As shown in Figure 159, TUN objects provide an optional generic mechanism (Transaction\_Info) for provision of transaction-specific information without carrying the overhead in the wireless transfer.

Both Connection\_Info and Transaction\_Info can be used simultaneously.

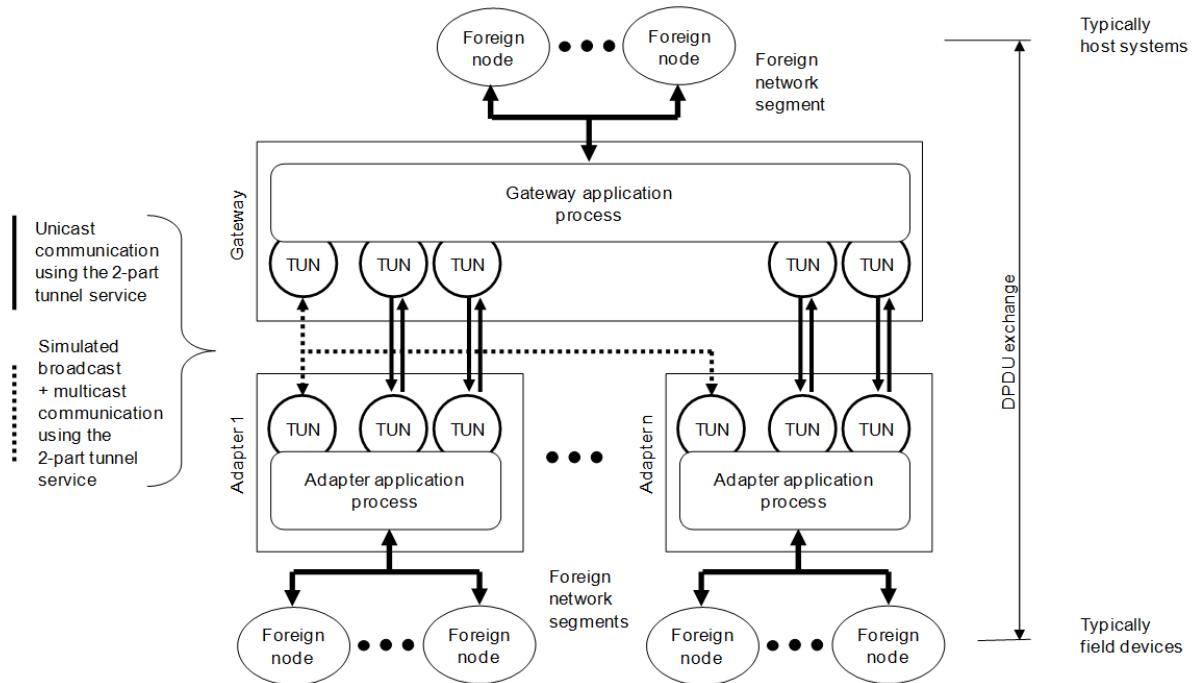


**Figure 159 – `Transaction_Info` usage in protocol translation**

### 13.3.1.10 Interoperable tunneling mechanism

#### 13.3.1.10.1 Overview

This clause defines a communication mechanism for foreign network nodes to communicate across a wireless network via gateways and adapters. This mechanism enables vendor-independent development of interoperable gateways and adapters by implementing a restricted subset of the communication features defined within this standard. The interoperable communication is achieved by the use of a constrained tunneling mechanism. The gateways and adapters serve to interconnect two or more foreign network segments by bridging foreign protocol DPDUs through the wireless network as depicted in Figure 160. The gateway and adapter application processes utilize the application layer tunnel objects and services for the exchange of foreign unicast DPDUs and foreign broadcast/multicast DPDUs. The mechanism by which the gateway and adapter application processes exchange these DPDUs with foreign nodes is not specified by this standard.



**Figure 160 – Interoperable tunneling mechanism overview diagram**

#### 13.3.1.10.2 Tunnel object placement

One or more foreign network nodes, individually addressable by a unicast DPDUs, may exist behind a gateway or an adapter. A foreign network node behind a gateway or adapter may require communication with an associated foreign network node behind another gateway or adapter.

For each associated gateway and adapter, a tunnel object shall be disposed and configured to carry foreign broadcast and multicast addressed DPDUs, one for a first associated foreign network segment and one for each additional associated foreign network segment.

For each associated foreign network node pair, a pair of tunnel objects shall be disposed and configured to carry unicast addressed DPDUs, one in the gateway or adapter for a first associated node and one in the gateway or adapter for a second associated node.

#### 13.3.1.10.3 Tunnel object configuration

Tunnel operation is controlled as described in Clause 12. Tunnel objects are configured through attribute settings. Changes to the configuration are required to be correctly sequenced by setting the Protocol attribute and monitoring the Status attribute.

The unicast tunnel object pairs shall be configured as follows:

- The Flow\_Type attribute shall be configured for a 2-part tunnel
- The Array of Tunnel endpoints attributes shall be configured for a single address element, where each tunnel object in the pair addresses the other tunnel object in the pair
- The Connection\_Info[ ] and Transaction\_Info[ ] attributes shall not be used
- The Update\_Policy, Phase, Period and Stale\_Limit attributes shall not be used

For unicast tunnel objects, the Foreign\_Destination\_Address attribute of each local tunnel object shall be set to the DPDUs address of the associated foreign device behind the remote gateway or adapter and the Foreign\_Source\_Address attribute of each local tunnel object shall be set to the DPDUs address of the associated foreign device behind the local gateway or adapter.

The tunnel objects in the broadcast/multicast tunnel object set shall be configured as follows:

- The Flow\_Type attribute shall be configured for a 2-part tunnel.

- The Array of Tunnel endpoints attributes shall be configured for one or more address elements, where each tunnel object in the set addresses all other tunnel objects in the set.
- The Connection\_Info[ ] and Transaction\_Info[ ] attributes shall not be used.
- The Update\_Policy, Phase, Period and Stale\_Limit attributes shall not be used.

For broadcast/multicast tunnel objects, the Foreign\_Destination\_Address attribute and the Foreign\_Source\_Address attribute shall be set to an equal value.

The usage of the Foreign\_Source\_Address attribute and the Foreign\_Destination\_Address attribute enables gateways and adapters using the interoperable tunneling mechanism to be configured strictly by configuration of the tunnel objects.

Associated gateways and adapters shall send and receive foreign DPDUs from either identical versions or interoperable versions of the same foreign protocol. To enable the run state after the other attributes are configured, the Protocol attribute shall be configured last and shall be configured to the same value in all tunnel objects associated with all related foreign network segments on the DL subnet. The final Protocol attribute value shall be set as defined in Annex K. The gateway and adapter application processes shall report tunnel object Status = 2 (configuration failed) if an attempt is made to configure a tunnel with an unsupported Protocol.

NOTE A compatible foreign protocol should be able to accommodate the timing imposed by the wireless mechanisms, either inherently or by configuration. Exchange of foreign DPDUs may not be the most efficient tunnel method, but it assures that sufficient information is available to process the packet within gateway and adapter application processes. It also assures multiple vendors convey the same information between gateway and adapter application processes. It is also assures that sufficient information is available within gateway and adapter application process to link client/server requests and responses. Addressing is also carried and enables multiple foreign network devices to sit behind each gateway or adapter.

#### **13.3.1.10.4 Tunnel operation**

Foreign network DPDUs may be delivered to the gateway and adapter application processes in one of two ways, either through a tunnel object or from a foreign source outside of the wireless network. The outside source will typically be a wired network (and its associated protocol stack) attached directly or indirectly to the gateway or adapter. Alternatively, the PDUs may be generated by software or firmware interacting with (or embedded in) the gateway or adapter application process directly. In either case, the tunneled PDU exchange between gateways and adapters shall remain identical.

The gateway and adapter application processes shall examine the foreign network DPDU destination address prior to forwarding the PDU over the wireless network. DPDUs without a known destination that is reachable through the tunnels shall not be forwarded.

Gateways and adapters application processes shall forward foreign protocol unicast DPDUs to DPDU address destinations that are reachable through a linked pair of unicast tunnel objects.

Gateway and adapter application processes shall forward valid foreign protocol broadcast and multicast DPDUs through the broadcast/multicast tunnel that exists within each associated gateway and adapter, distributing the same DPDU to one or more destinations. The PDU shall not be echoed back to the source.

The gateway and adapters application processes shall utilize multicast group establishment PDUs from within the foreign protocol, where such PDUs exist, in order to limit distribution scope.

NOTE Since generation of multiple copies of the same message is almost certain to occur, the foreign protocol is required to tolerate timing skew.

#### **13.3.1.10.5 Efficient operation**

It is recommended that foreign protocols that are using the interoperable tunneling mechanism reduce PDU exchanges to the minimum that is acceptable to the foreign protocol and its applications. This is accomplished by extending update periods and timeouts for periodic update. This is also accomplished by elimination of redundant transfer of static information by maintaining local copies.

### 13.3.2 Bulk transfer

Large item transfer shall be accomplished through upload/download objects (UDOs), as shown in Figure 161. Transfers are typically used for firmware updates, transfer of large sample buffers, and general configuration. One UDO represents a single item that can be transferred in either direction (uploaded or downloaded) to/from another application. The item to be transferred exists at the location of the UDO. Interface objects (IFOs) act as clients to initiate transfers. The transfer protocol provides buffering, flow control, and guaranteed and in-order delivery. Protocol translators have access to the UDO through the GSAP.

Items are associated with a string that can be used to encode item specific identification and revision information. Asset management systems can be constructed to monitor revisions for regulated industries and to backup and restore items generically, without knowledge of the item content. Protocol translators may also transfer large items via foreign protocols through tunneling, but this precludes protocol independent asset management. It is recommended that items to transfer such data be exposed through UDOs.

End applications are expected to understand the content of the transferred item and how to apply it. Provisions exist (depending on device capabilities) to request utilization of the item (possibly altering run-time behavior) and for storage of the item in non-volatile memory.

The UDO and the upload and download bulk transfer protocol are described in 12.15.2.4.

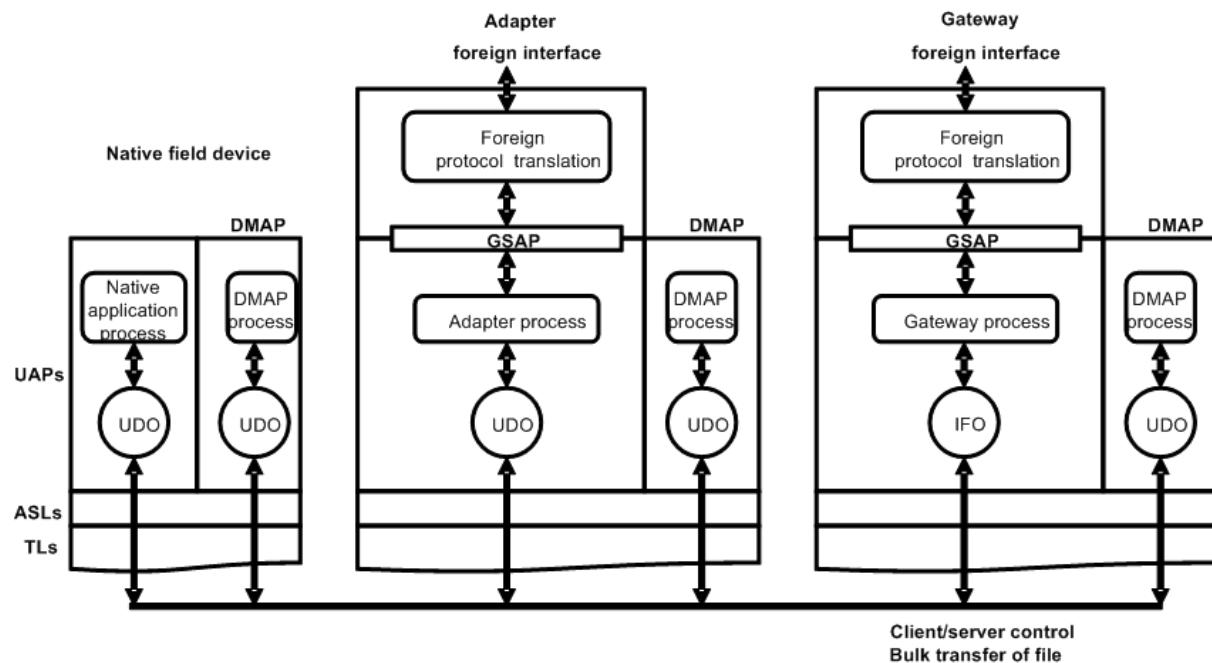


Figure 161 – Bulk transfer model

### 13.3.3 Alerts

Alerts may be generated by many of the objects defined by this standard. Some objects reside within device UAPs, while others reside in the DMAP as management objects for each layer.

Alerts within a device are consolidated within the alert reporting management object (ARMO). Each device has a single ARMO that resides within the DMAP. All alerts within a device are conveniently consolidated in this single location.

The ARMO in each DMAP is responsible for reporting alerts through an AlertReport service to an alert receiving object (ARO). The ARO acknowledges alert receipt through the AlertAcknowledge service. This transfer occurs independently of the actual processing of the alerts.

Alerts fall into four categories:

- Process;
- Device;
- Network; and
- Security.

Each category can be delivered by an ARMO to a different ARO. Thus, a single ARO might collect all process alerts across an entire network, or a set of AROs can be used, with each ARO only collecting a single category of alerts. If each ARO collects only one type of alert, then collection of all alerts requires four AROs.

The gateway contains one or more AROs that allow collection, reporting, and management of alerts.

Protocol translators have access to and can manage alerts through the GSAP, as shown in the alert model in Figure 162. Subscription services allow alert selection through the GSAP.

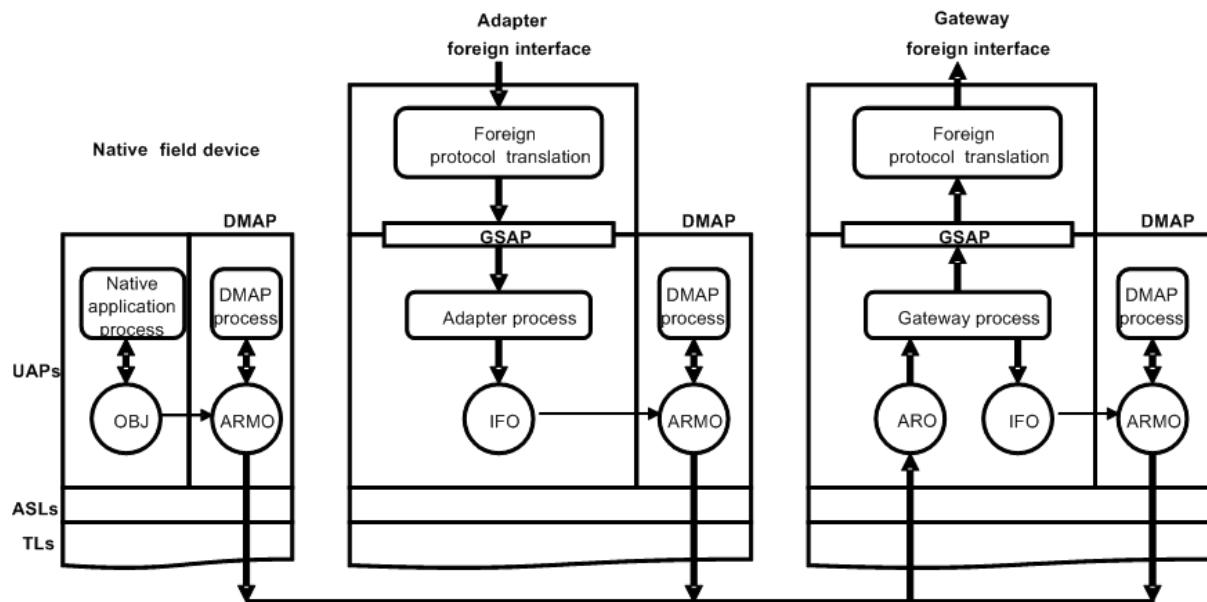


Figure 162 – Alert model

Alerts fall into two classes, alarms and events. Events are informational and generate event messages through the GSAP. Alarms have states and require alarm-specific actions to clear the alarms. Client/server messaging is typically used to perform these actions.

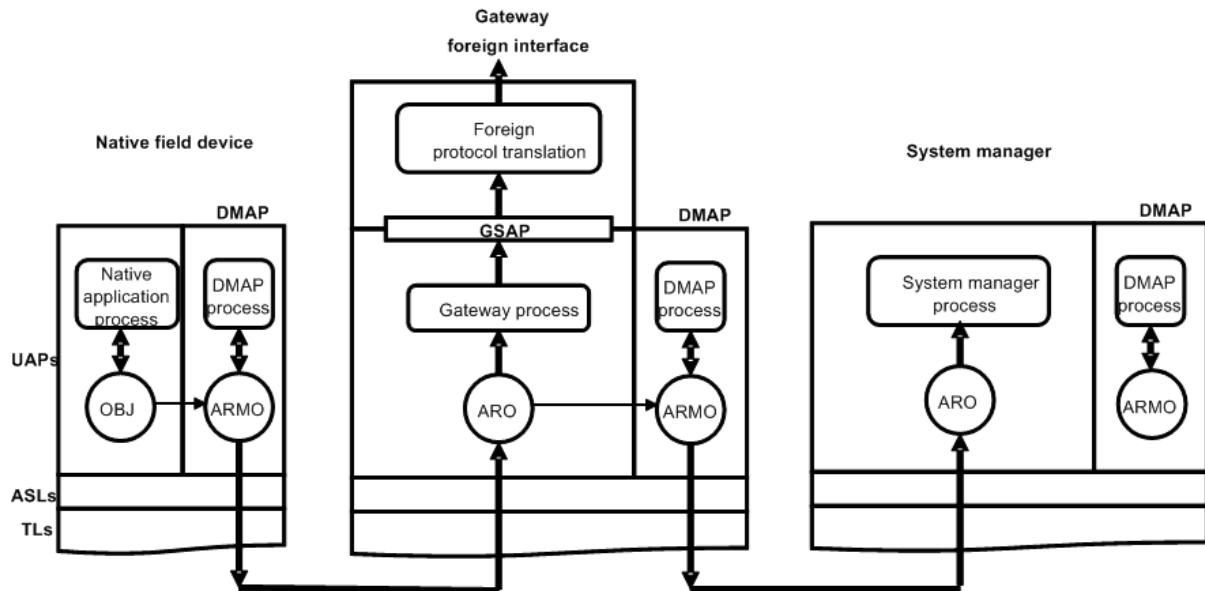
The gateway and the adapter applications are also able to generate native alerts from IFO instances. This allows protocol translators to generate alerts within the context of a standard alert management system.

In certain circumstances, the state of alerts may be lost at the ARO, such as when a gateway is reset or replaced. In such case, the original ARMOs will no longer contain information about events, but will maintain state information related to alarms. An alarm recovery procedure can be initiated in order to recover the system alarm state.

This standard does not support multicast alerts. As a result, the same alerts cannot be routed to both the gateway and the system manager if they are not physically co-located. Network and security alerts are currently sent to the system manager by default. Process and device alerts are sent to the gateway by default.

The alert model does not support multicast alerts. Network alerts and security alerts are potentially useful in a gateway for transformation into generic foreign protocol error messages. The system manager is the default destination for these alerts. The ARO in the gateway, when configured to collect alerts for network and security purposes, shall be capable of

reposting the alerts through the local ARMO to the system manager. This is illustrated in Figure 163.



**Figure 163 – Alert cascading**

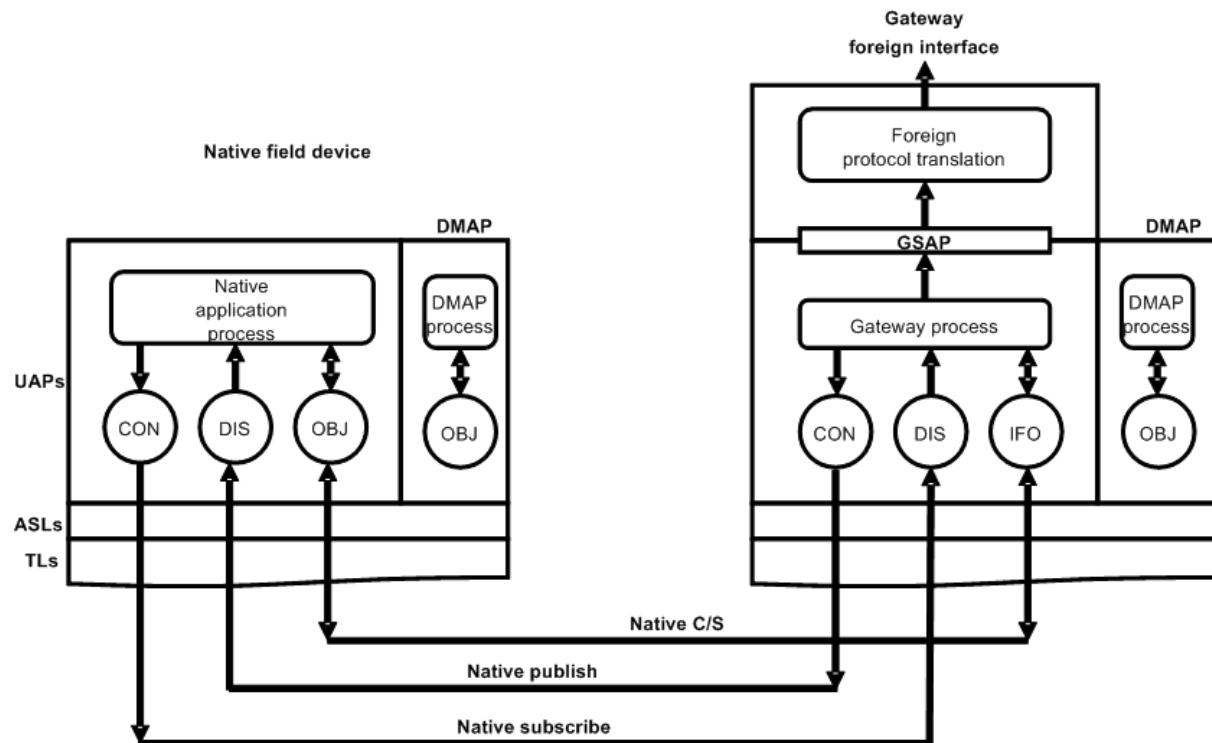
#### 13.3.4 Native publish/subscribe and client/server access

This standard provides publish/subscribe and client/server services via the ASL that shall be used to interact with application-specific native objects.

For publish/subscribe, the concatenation (CON) and dispersion (DIS) objects shall be utilized as endpoints.

For client/server services, two object endpoints are required in order to utilize these services. The IFO shall act as one endpoint for these services within gateways. Any other application or management object within the system can act as the other endpoint.

As shown in Figure 164, utilization of these objects allows protocol translators to integrate simple devices that do not include legacy protocols.



**Figure 164 – Native P/S and C/S access**

Within a gateway, the CON and DIS objects shall provide buffered message behavior for change of state operation.

Within a gateway, the IFO shall provide buffered message behavior as described for the 4-part tunnel messaging between tunnel objects for C/S read services. The IFO shall utilize the attribute classification to determine buffering behavior. Non-bufferable attributes shall not be buffered. Constant attributes shall be buffered. Static and dynamic attribute buffering shall be determined by application requirements.

The protocol translator shall utilize native addressing (Network\_Address, Transport\_Port, OID, and attribute identifier) to identify native messages.

**NOTE** Tunneling assumes that foreign protocol messages are transferred between endpoints. As such, foreign addresses are associated with the messages and utilized for teardown and reconstruction of the messages in order to avoid transfer. No such assumption is made for native messaging, where a one-to-one message flow is less likely to exist.

### 13.3.5 Time management

Host time may be propagated through a gateway to a wireless system, giving the host system and the field devices the same sense of time (within tolerances). This enables the host time to be used for purposes such as uniform alert timestamping and sequence of event determination that spans wireless and wired devices connected to the host. Without periodic synchronization to host time, the wireless system will drift, thus periodic adjustment capability is desirable. Both the host and wireless system may be synchronized to a common external source such as a GPS derived timesource.

To propagate host time, a gateway may perform periodic synchronization of time in an attached DL subnet time to an external source by requesting time changes through a DLMO.

Protocol translators within a gateway shall access time management functions through the GSAP services. Protocol translators shall be responsible for accessing external time sources and converting protocols and time formats. Network time is represented in TAI format, as described in 5.6.

A DL configured as a clock master is used to propagate time synchronization information to an attached DL subnet, as described in 6.3.10.3. Each node contains a DMO within its DMAP. The DMO contains attributes `DL_Subnet_Clock_Master_Role` and `DL_Subnet_Clock_Repeater_Role` that control the ability of a node to be a clock master. Allocation of the clock master role is coordinated with the system manager. The device registers its ability to be a time source during the join process.

The DLMO contains an attribute called `TaiTime` that reports the current time and another attribute called `TaiAdjust` for adjusting the time. The DLMO shall be used to adjust the time of the DL subnet.

One or more DLs may be associated with a gateway. In one implementation, the DLs are integrated within the gateway. In another implementation, the DLs are within backbone routers and separated from the gateway, adding indeterminate delays. Each implementation shall consider the implications of delay associated with access of DL objects to perform synchronization.

### **13.3.6 Security**

Sets of wireless devices are related to a foreign host via a gateway. The gateway and the wireless devices are expected to belong to a common security group. Security for this group may be established by MAC or transport layer security configuration, or both as described in Clause 7. Establishment of common security settings is a prerequisite for communication between protocol translation communication endpoints.

Common foreign fieldbus protocols do not have security capabilities. This does not preclude extension of secured protocols into this standard's domain. It is the responsibility of foreign protocol translators in both gateways and adapters to act as trusted applications in the extension of foreign protocol security from end-to-end. This can be achieved by utilization of native security or through tunneled exchanges.

### **13.3.7 Configuration**

The gateway configuration is based on management attributes that extend the basic UAP management object (UAPMO), as described in Table 419.

**Table 419 – UAP management object extended attributes**

<b>Standard object type name: UAP management object extended attributes</b>				
<b>Standard object type identifier: 1</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of attributes behavior</b>
Max_Devices	11	Maximum number of devices supported by gateway	Type: Uint16	Implementation dependent value set by gateway depending on resources
			Classification: Static	
			Accessibility: Read only	
			Initial default value: 0	
			Valid value set:0 – 0xFFFF	
Actual_Devices	12	Current number of devices connected to gateway	Type: Uint16	Increases and decreases based on devices in communication with the gateway
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set:0 – 0xFFFF	
Max_Lesees	13	Maximum number of leases supported by the gateway	Type: Uint16	Implementation dependent value set by gateway depending on resources
			Classification: Static	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set:0 – 0xFFFF	
Actual_Lesees	14	Current number of leases for devices connected to the gateway	Type: Uint16	Increases and decreases based on leases. Device complexity will determine the number of leases required
			Classification: Dynamic	
			Accessibility: Read only	
			Initial default value: N/A	
			Valid value set:0 – 0xFFFF	

### 13.3.8 Provisioning and joining

A gateway is a network device as described in this standard and shall be provisioned utilizing the generic methods described in this standard

A gateway that communicates to DL subnets through backbone routers shall provide a method to configure the gateway to communicate to a specific DL subnet and to specific devices within that subnet through a specific backbone router.

NOTE 1 Nothing precludes more dynamic implementations. For example, a load-sharing algorithm may assign devices to the best BBR found. Gateways and BBRs may discover each other. Redundancy may be automatic where subnets overlap. Static mapping is a minimum requirement.

A gateway that communicates to one or more DL subnets through backbone routers shall include a method to configure the gateway to communicate with at least one system manager, where the system manager may reside:

- In the gateway;
- On a backbone that the gateway can use for communication; or
- Within a DL subnet that the gateway can use for communication.

A gateway is a network device (containing an AL and network address) as described in this standard and shall join the network following the join methods described in this standard.

NOTE 2 Several variations are possible. A gateway may join through sending an internal request to a co-resident system manager, or send a join request through a local PhL, or may utilize a backbone router's PhL indirectly, or may send the join request across the backbone to a system manager.

## 14 Provisioning

### 14.1 General

A device conforming to this standard is considered provisioned when the device has the information required to communicate with a target network and initiate a join request to the system manager/security manager of the target network. In this document, a target network is defined as a network the device is being provisioned to join. The information required to initiate the join request includes both security (trust-related) information and network-related information. The provisioning clause specifies over-the-air provisioning procedure and message format where the Type A field medium is used and out-of-band message formats where the Type A field medium is not used to provision the trust related and network related information.

Over-the-air provisioning uses join processes to set up a connection between the provisioning device and the device being provisioned. The join process is described in 6.3.9.2 and follows two optional paths, one defined for a device joining with trust-related information based on symmetric key, and another defined for a device joining with trust-related information based on asymmetric (public/private key pair) key. Out-of-band provisioning may not use join processes and provision the information via another wired or wireless means.

The goal of the provisioning process is to provide enough information so that one of these paths can be taken by the device.

The provisioning process involves a device that implements the provisioning role by providing the network-related and trust-related information to the new device. During provisioning, the operator can use the provisioning device, acting as a proxy for the system manager, to decide if a new device should be connected to the network or not, with information from security manager. In another example, a copy of the list of allowed devices can be obtained from the security manager, allowing the provisioning device to make a local decision. When the target network is a secure network both trust related and network related information needs to be provisioned; for unsecured networks the default key ( $K_{\text{global}}$ ) is used as the trust-related information. Once a device is provisioned, it is ready to join the target network. Thereafter, usually without human intervention, the security manager of the target network may either accept or reject the join request to the target network from the device based on the provisioned information.

**NOTE** In this document, various aspects related to installation of the trust-related and network-related information in a device, conveyance of this information to the security manager, and establishment of trust are described in different clauses. Installation of the trust-related and network-related information is described in this clause. Conveyance of the information to the security manager is described in Clauses 9 and 10. Establishment of trust is described in 7.4.4.3.2.

### 14.2 Terms and definitions

The following terms are defined for devices with various roles or states:

- **Device to be provisioned (DBP):** A device that needs to be provisioned, or is in the process of being provisioned. The device may be missing all or part of the information required to join a network.

**NOTE** 1 The device could contain old information relating to a network, but may be required to be provisioned with new information.

- **Target network:** The network that the DBP is being provisioned to join.
- **Provisioning device (PD):** A device that implements the role of provisioning another device to allow that device to join the target network. A PD need not be a device implementing only the provisioning role; rather, it could be:
- **The system manager/security manager of the target network.**

**NOTE** 2 The system manager/security manager role may be distributed to a set of devices in the target network.

- A device, such as a handheld device containing a system manager/security manager, that uses the protocol suite specified by this standard to provision the DBP through a separate, temporary mini-network.
- A device that uses out-of-band (OOB) communication, such as infrared, near field communications (NFC), or plugs, to provision the DBP. This OOB communication is outside the scope of this standard.
- Default network: The network whose network identifier is 1.
- Provisioning network: A network formed between the PD and the DBP. If the PD is a handheld, then the provisioning mini-network is the network formed between the handheld and the DBP. If the provisioning device is the security manager of the target network, then the provisioning network may be a separate logical network on the target network itself.
- Join key ( $K_{join}$ ): An AES join key used to join a secure target network. The value of key  $K_{join}$  is intended to be secret, and thus is intended to offer data confidentiality. The value of key  $K_{join}$  is updated during provisioning to a new value that is known only to the target network security manager and the device.<sup>9</sup>
- Default join key ( $K_{join} = K_{global}$ ): An AES join key with a published value. The value of  $K_{global}$  is not intended to be a secret; its value is well known. It therefore does not offer data confidentiality, but does help improve data integrity. Its purpose is to establish connectivity between devices compliant to this standard that do not share a secret join key. Such connectivity is needed for:
  - Over-the-air (OTA) provisioning of target network related information;
  - OTA reading of device identity and configuration settings;
  - OTA authentication of device credentials; and
  - OTA updating of join key  $K_{join}$  (the latter two steps using asymmetric cryptography).

The value of the default join key shall be  $K_{global}$ , defined in 7.2.2.2.
- Open join key ( $K_{join} = K_{open}$ ) A published non-secret value for the join key ( $K_{join}$ ). This special value for the join key is used to join a provisioning network so that certain OTA symmetric-key only provisioning methods can be facilitated. The actual value for this key is defined in 7.2.2.2. See 3.3.1.3 and refer to OPEN encoding, as yet non-existent.
- Physical and logical networks: A physical network is a set of physical devices that communicate with each other, possibly through multiple hops. A logical network is a network instance that runs on the physical network. One physical network may support multiple logical networks. Logical networks have different individual priority and security properties. For example, the target network and the provisioning network are two logical networks that exist on a physical network.
- Idle state: Device state that is not actively participating in the wireless network,
- Provisioning state: The device is in the provisioning phase.
- Provisioned state: The device received enough information to join target network, and got the DMO.Join\_Command=1.
- Factory defaults: The default configuration of a field device as it comes out of a manufacturing facility. The default configuration has  $K_{global}$  and  $K_{join}$  equal to their default values, and OTA provisioning allowed. An operational device may be reset to factory default, either by the system/security manager when it is part of a secured network or by OOB means using a provisioning device. Factory defaults for the provisioning process are summarized in Table 420. Only the system manager shall have the authority to reset a device to factory defaults, externally.

NOTE 3 This specification does not preclude devices that do not allow reset to factory defaults.

---

<sup>9</sup> Appropriate mechanisms are provided so that the protocol suite defined by this standard cannot be used to read the current value of the join key from a device. Note that the secrecy of join keys cannot be enforced by this standard.

### 14.3 Provisioning procedures

All field devices compliant with this standard shall implement a standard object called the device provisioning object (DPO). Attributes of the DPO in the DBP shall specify the information required to initiate a join request to the target network. The device shall retain all attributes of the DPO through a power cycle or battery replacement. The device provisioning object is described in detail in 14.9.1.

NOTE 1 This specification does not preclude that the system manager can have the DPO, for example, store the security manager EUI-64.

PDs shall implement a device provisioning service object (DPSO) that contains information intended for the DBPs that are serviced by the PD.

Provisioning involves setting the attributes of the DPO. The attributes in the DPO contain both network-related and trust-related information. These attributes can be set via three different means:

- They may be pre-installed at the device manufacturer;
- They may be set using OOB means; or
- They may be set by a PD using a provisioning network. The PD in this network acts as a proxy for security manager/system manager of a target network to provide the trust-related and network-related information for that target network.

All devices complying with this standard shall support the formation of the provisioning network using only the full protocol suite defined by this standard (PhL, DL, NL, and TL), i.e., not requiring any other mechanism. However, this standard does not disallow provisioning by OOB communication means.

When using the Type A field medium (5.2.6.2.2) in the provisioning network, standard messages and header formats shall be used to set the attributes of the DPO. The DPO defines a set of default read-only attributes for the formation of the provisioning network or an insecure network. The default attributes include default published symmetric keys ( $K_{join} = K_{global}$  and  $K_{join} = K_{open}$ ), a default sub-network identifier, and a default set of frequencies. Since this set of default attributes is known and contained in the DPO of all devices conforming to this standard, the attributes provide a means for all devices to join a provisioning network.

The DPO provides an attribute (DPO.Allow\_Provisioning) so that access to the attributes of the object via the default open instance can be either allowed or blocked.

NOTE 2 Some devices may implement an external mechanism (i.e., a switch) that will lock the provisioning state (either blank or provisioned) of the device, to minimize battery consumption and also to minimize the likelihood that a rogue PD will re-provision a device.

### 14.4 Pre-installed symmetric keys

The formation of a provisioning network is not a necessary step for provisioning; the trust-related information can be pre-installed in a device. For example, it is possible for a user to delegate (partly) the provisioning of devices to a device manufacturer or to a third party. A device manufacturer may pre-program secret symmetric join keys into devices, and may supply this same secret symmetric join key data to the user so that the data can be loaded to the system manager/security manager of the target network. Alternatively, the user may stipulate to the device manufacturer what symmetric key shall be loaded. In this case, the DPO of a device shall be pre-installed with target network information and target symmetric join key  $K_{join}$ . Depending on the application, two or more devices may share the same secret information. Devices with pre-installed trust information and target network information can proceed directly to the join process.

Devices with pre-installed trust-related information but no target network-related information shall have the option to be provisioned with necessary network information. This facilitates the device to receive advertisements from the target network on the correct frequencies, to expedite the join process, and to present join requests only to the target network. If the

network-related information is not provisioned, a device may scan for advertisements from networks in its vicinity using the default network settings.

#### **14.5 Provisioning using out-of-band mechanisms**

Devices without pre-installed symmetric keys need to be authenticated and then provisioned with trust information. As noted earlier, this can be accomplished either through the provisioning network Type A field medium over-the-air or through OOB mechanisms.

OOB communication means include, but are not limited to, infrared, wired connectors, memory cards, keyboards on devices, NFC, and plugs. The mechanism of OOB communications is outside the scope of this specification. The attributes of the DPO that specify the joining to the target network should be set to the same values regardless of the means used (over-the-air or OOB).

#### **14.6 Provisioning networks**

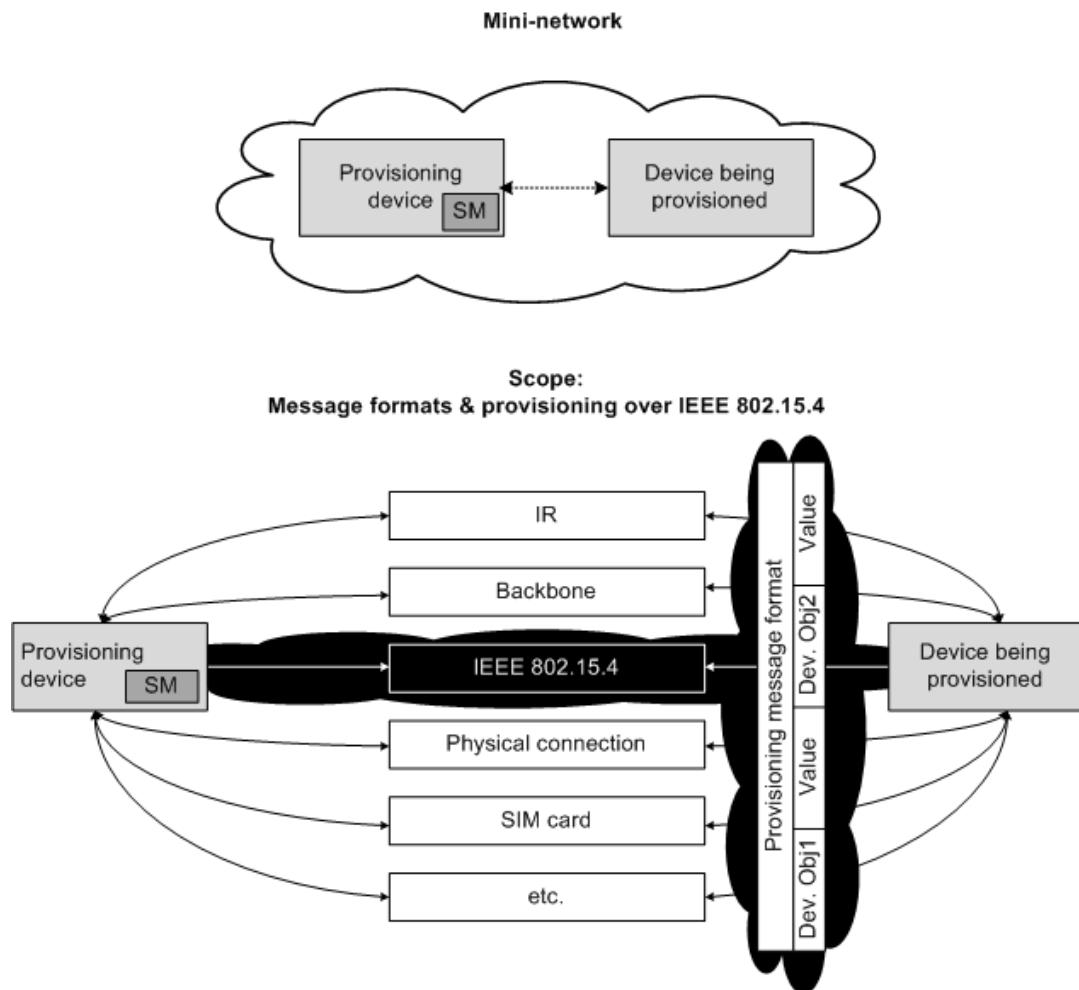
##### **14.6.1 General**

In addition to OOB-provisioning and factory pre-provisioning, this standard defines the formation of a standard network for provisioning devices over-the-air (OTA) using the Type A field medium. The default symmetric join key ( $K_{\text{global}}$ ) or open symmetric join key ( $K_{\text{join}} = K_{\text{open}}$ ) may be used as the trust-related information for the formation of the OTA provisioning network. The default join key ( $K_{\text{global}}$ ) is used for formation of the provisioning network to obtain target network-related information and target network join key and for devices with asymmetric cryptographic capability. The join key ( $K_{\text{join}} = K_{\text{open}}$ ) is used for the formation of a provisioning network where both trust-related and network related information is provisioned over the air. This form of provisioning is insecure and by default system managers and provisioning devices shall not allow joining with this join key.

A PD that has asymmetric cryptographic capabilities distinguishes the method with the key used to generate the MIC in the `Security_Sym_Join().request`. In the PD, the MIC generated by the device joining the default network needs to be validated a maximum of twice – one for  $K_{\text{open}}$  and the other for  $K_{\text{global}}$ . If security manager detects that  $K_{\text{global}}$  is used for the MIC, the DBP shall be provisioned using asymmetric cryptography. Otherwise, the DBP shall be provisioned using the open symmetric key).

The provisioning network can either be an isolated mini-network formed with a handheld device, or it can be a separate logical network on the target network itself. In the latter case, connectivity from the DBP to the advertising router is open but connectivity further on, from that advertising router to the system manager, is protected by existing session and thus secured. If the logical provisioning network is on the target network, the application objects of the system/security managers on the target network and the logical provisioning network (e.g., DPSO) can communicate with each other within the same device.

Figure 165 illustrates the provisioning (mini-)network.



**Figure 165 – The provisioning network**

OTA provisioning uses a PD that can be either:

- A handheld that forms an isolated mini network with the DBP. This handheld has its own system/security manager and an advertising router functionality; or
- The system manager / security manager of the logical provisioning network on the target network.

NOTE A mini-network is formed, and thus the functions of system and security manager are required in the PD to perform the join process, if it is used for OTA. If the device is used for OOB, then since the DBP does not perform the join process, the PD does not need to have the full functionality of system and security managers.

#### 14.6.2 Provisioning over-the-air using asymmetric cryptography

DBPs that are capable of performing asymmetric cryptographic calculations shall use the default join key ( $K_{\text{global}}$ ) to join a provisioning network. The DBP receives advertisements whose subnet ID = 1 from nearby advertising routers and initiates a join request using default symmetric key. A successful join process results in the PD and the DBP having established a contract for further communication. The PD then uses standard application layer primitives (such as read and write) to transfer the network-related information contained in its DPSO to the DPO of the DBP.

For provisioning the trust-related information the PD interrogates the DBP; i.e., it reads its credentials (e.g., DPO.PKI\_Certificate or multiple DPO.PKI\_Certificates; see Annex G), and sends those credentials to the security manager. The security manager of the provisioning network checks the credentials of the DBP and validates the authenticity of the DBP through a challenge-response mechanism. The security manager/system manager may ask for further confirmation from the user through a GUI to provision the DBP. Once accepted, the system manager provides the DBP with the secret AES join key so that the DBP can join the target network immediately or possibly at a later time, using that join key. When this new key is

transmitted over the air, it shall be encrypted by the asymmetric key of the DBP, which is part of its certificate, so that it cannot be intercepted while in transit.

Security managers conforming to this standard are not required to have asymmetric cryptographic capabilities; hence, some security managers may not be able to accept or provision devices with asymmetric cryptographic capabilities. When the DBP joins the provisioning network using  $K_{\text{global}}$ , security managers and PDs not capable of asymmetric cryptographic calculations shall not transmit the trust-related information to the DBP.

**NOTE** In addition to a high level of security, asymmetric cryptographic modules and certificates provide a convenient and easy means for devices to establish communication with the security manager of the target network and to be provisioned without the use of additional tools. It is recommended that manufacturers of security managers without asymmetric cryptographic modules provide adequate means (e.g., memory, processing power or additional peripherals, etc.) to upgrade such security managers, upon user request, to be able to accept and manage devices with asymmetric cryptographic modules.

#### **14.6.3 Provisioning over-the-air using an open symmetric join key**

This standard allows PDs to provision devices that do not have asymmetric cryptographic capabilities to be provisioned over the air. For this purpose, a well-known open symmetric join key ( $K_{\text{join}} = K_{\text{open}}$ ) is used. The security manager in the PD shall not permit OTA provisioning with open symmetric key,  $K_{\text{open}}$ , by default. The provisioning network is not secure, since it uses a published open key and join key for target network and thus can never be protected securely.

**NOTE 1** In OTA provisioning with  $K_{\text{global}}$ , the security information (i.e., join key) is encrypted with an asymmetric key while transmitting.

**NOTE 2** The use of an open symmetric join key for provisioning is not a secure procedure. An eavesdropping device may be able to obtain the join keys to the target network and pose a security risk when this provisioning procedure is used. This provisioning procedure should only be used in applications where the security risk is minimal and the user is either not concerned or has taken sufficient measures to avoid eavesdropping. Such exposure can be avoided by using asymmetric crypto-based provisioning or OOB provisioning.

The use of the open symmetric join key provisioning is optional. DBPs may be configured not to use OTA provisioning with this key. By default, security managers and PDs shall reject join requests from all devices that send join requests using the open symmetric join key. Security managers and PDs need to be configured to accept join requests using the open join key. Some security managers and PDs may not allow configuration to this optional feature.

The device that joins a provisioning network using this join key may be provisioned with the join key for a target network by the PD. However, once provisioned with a new join key for the target network, the device shall not be allowed to use the open symmetric join key unless the device is reset to factory defaults. The only means for the device to reuse this key for provisioning is to reset the device to factory defaults.

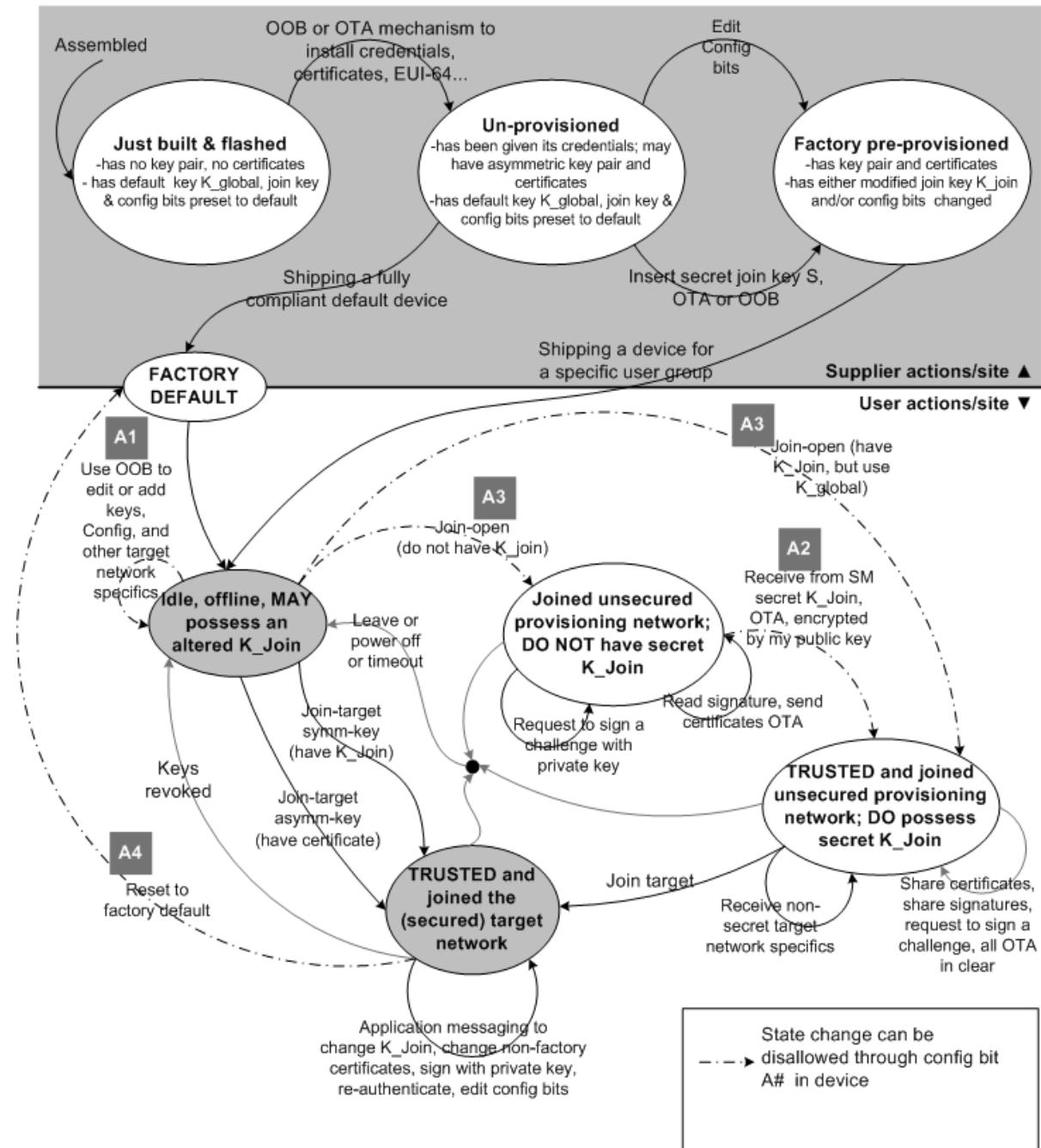
The provisioning procedure using the open symmetric join key can be used either in the provisioning (mini-)network or through a separate logical network on the target network. The DBP receives an advertisement from a provisioning network and a standard join request is sent using symmetric key ( $K_{\text{join}} = K_{\text{open}}$ ). If the request is accepted, the DBP joins the provisioning network and a contract is established between the DBP and PD. Application-level read/write primitives and methods are available to the PD to provision the trust-related and network-related information; this includes, for example, provisioning the target join key using the DPO.Write\_Join\_Key method.

The provisioning (mini-)network can also be used for device configuration. Since a contract has already been established, the PD may also use the network (either OTA or OOB) to configure the DBP.

### **14.7 State transition diagrams**

The options discussed thus far for provisioning are shown below in state transition diagrams.

Figure 166 depicts the state transitions relevant for provisioning through the life cycle of a field device. The diagram depicts states at a manufacturing site and a user site.



**Figure 166 – State transition diagrams outlining provisioning steps during a device life cycle**

A field device that is newly manufactured transitions to the un-provisioned state when its identity (e.g., EUI-64) and credentials are provided to it. In this state, the device has the default settings as defined in Table 420.

**Table 420 – Factory default settings**

<b>Attribute</b>	<b>Description</b>	<b>Default value</b>
Default SYM join key K_global	The AES symmetric join key used to join a default network	ISA(space)100(null). Actual value is specified in 3.3.1.3.
Open SYM join key (K_join = K_open)	The open AES symmetric join key used to join a provisioning network, then to receive new target join keys	OPEN(null)(null)(null)(null) Actual value is specified in 3.3.1.4
Allow OOB provisioning (A1)	This configuration bit allows the use of OOB mechanisms for provisioning the device. This bit is irrelevant if the device does not have any OOB means for provisioning	1 = allowed
Allow asymmetric key based provisioning (A2)	This configuration bit allows the use of asymmetric crypto for OTA provisioning of K_join. This bit is irrelevant if the device does not support asymmetric cryptography	1 = allowed
Allow Default Join (A3)	This configuration bit allows the device to join a default network. Some devices may choose not to allow a default join at all	1 = allowed
Allow reset to factory defaults (A4)	This configuration bit allows execution of OTA commands that reset the device to the factory default configuration	1 = allowed

A device may be shipped with these default settings to the user site from the manufacturing site.

Alternatively, the device may be pre-provisioned for a particular user at the manufacturing site. When a device is pre-provisioned, the default settings of the device are changed. The device may be given a new target symmetric join key specific to a target network at the user site. In addition, any of the configuration bits (A1, A2, A3 and A4) may be changed. For example, the default network join and reset to factory defaults may be disabled (A3, A4 = 0). Such a device shall not be able to be provisioned through the open symmetric join key (K\_join= K\_open).

The device arrives at the user site either pre-provisioned or with factory defaults and is in the idle state.

At a user site, a device may be provisioned, using OOB mechanisms (A1 enabled), with AES join key and network-related information for joining the target network. Alternatively, the device may already have preinstalled secret join keys and/or network-related information established by the device manufacturer. If the network-related information is not provisioned into the device at manufacturing, the device may join a provisioning network using the default symmetric join key (K\_global = ISA(space)100(null),, if A3 is enabled) and may be provisioned with network-related information using over the air mechanisms.

Devices that fail to join the target network using their provisioned information may seek to join a provisioning network (if A3 is enabled) using K\_global. After joining using the default join key (K\_global), the PD may use the Write\_Symmetric\_Join\_Key method to update the K\_join only if it is sent encrypted with the asymmetric key of the DBP.

If the device in an idle state does not have a valid installed AES join key and is allowed to join a default network, and A4 is enabled, the device shall start scanning for advertisements in order to reach a security manager/system manager of a default network in its vicinity.

If an advertisement is found and the device has asymmetric cryptographic capabilities and PKI certificates, it shall forward its credentials to the security manager associated with the advertising router. The advertising routers shall forward join requests to their security managers using an established contract that the advertising router has with the security manager/system manager.

When the security manager receives new device credentials, it first checks whether devices with those credentials are expected and authorized for the target network. This may be accomplished via lookup in pre-populated white lists with the EUI-64 of the individual device. The device credentials are used by the system manager to decide the CA (and its asymmetric key) to use in subsequent authentication steps.

If the device is authorized, then the authenticity of the credentials is checked by the system manager. The device credentials include the device certificate or multiple certificates. When utilizing the option of multiple certificates, the check on device data may<sup>10</sup> consist of two asymmetric crypto steps, one using the CA asymmetric key that is already present inside the security manager (the PD) to read the first certificate (termed the issuer certificate) and hence the issuer asymmetric key, followed by the second certificate (termed the device certificate) and hence the asymmetric key of the device, using the issuer asymmetric key. Once the asymmetric key of the device is obtained, a challenge/response mechanism (see 7.4.6) is used by the PD to establish the authenticity of the DBP.

A copy of data exchanged in preceding steps may be logged in public files in the PD for future audit purposes.

An optional step of soliciting user input to accept the device may occur before the device is accepted. A dialog on a human-machine interface (HMI) connected to the system manager may seek confirmation that the trustworthy device should be allowed to join the target network. This can be a yes/no dialog that asks if a specific device, with a specific authenticated identity, that is a member of a family of expected and deemed welcome devices, should indeed now be prepared for a secure join to the secured target network. When this optional step is implemented, and the user response is not received and no response is sent within the join response timeout period, the device will consider the join request as a failure.

If the device is authorized (present in the White List) and authentic, the PD generates a new key for the DBP, encrypts it using the DBP asymmetric key and transmits it to the DBP. A copy of that may be logged in public files in the PD for future audit purposes.

Failure in any of the steps above can be due to loss of connectivity, timeouts, or denial of join request from the DBP. Examples of the latter include a negative status on white lists, a mismatch while authenticating, or a reject from a dialog on an HMI. When it is clear that a DBP should be rejected for any of those reasons, an alert is generated by the security manager. No join response shall be sent back to the device indicating a join failure to the device.

If the DBP does not have asymmetric cryptographic modules but has the open symmetric join key, it can join a provisioning network with open symmetric join key ( $K_{join} = K_{open}$ ). The right to accept or reject provisioning of DBPs that use the open symmetric join keys ( $K_{join} = K_{open}$ ) rests with the PD. By default, the PD shall not provision devices that join with the open join key; however, the PD may be configured to provision such devices. If the PD is configured to allow open OTA provisioning, then the DBP will be provisioned with a new join key  $K_{join}$  for joining the target network. Once provisioned, the device shall not use the open key again unless it is reset to factory defaults (A4 is enabled).

Once provisioned, the device can proceed to join the target network with its provisioned information. As part of the join process, the device receives a master key, session keys, and DL keys, in addition to establishing a contract with the system manager/security manager of the target network, and normal operation of the standard secured network follows.

As part of the normal operation of a network, the system manager of the network may provision the device with sufficient information to join another network when the device leaves the current network. This process enables a device to join and leave multiple networks. Provisioning for another network using a current target network is accomplished as follows.

- 1) The DPSO in the current system manager retrieves network information and security keys from the system/security manager of the other network.

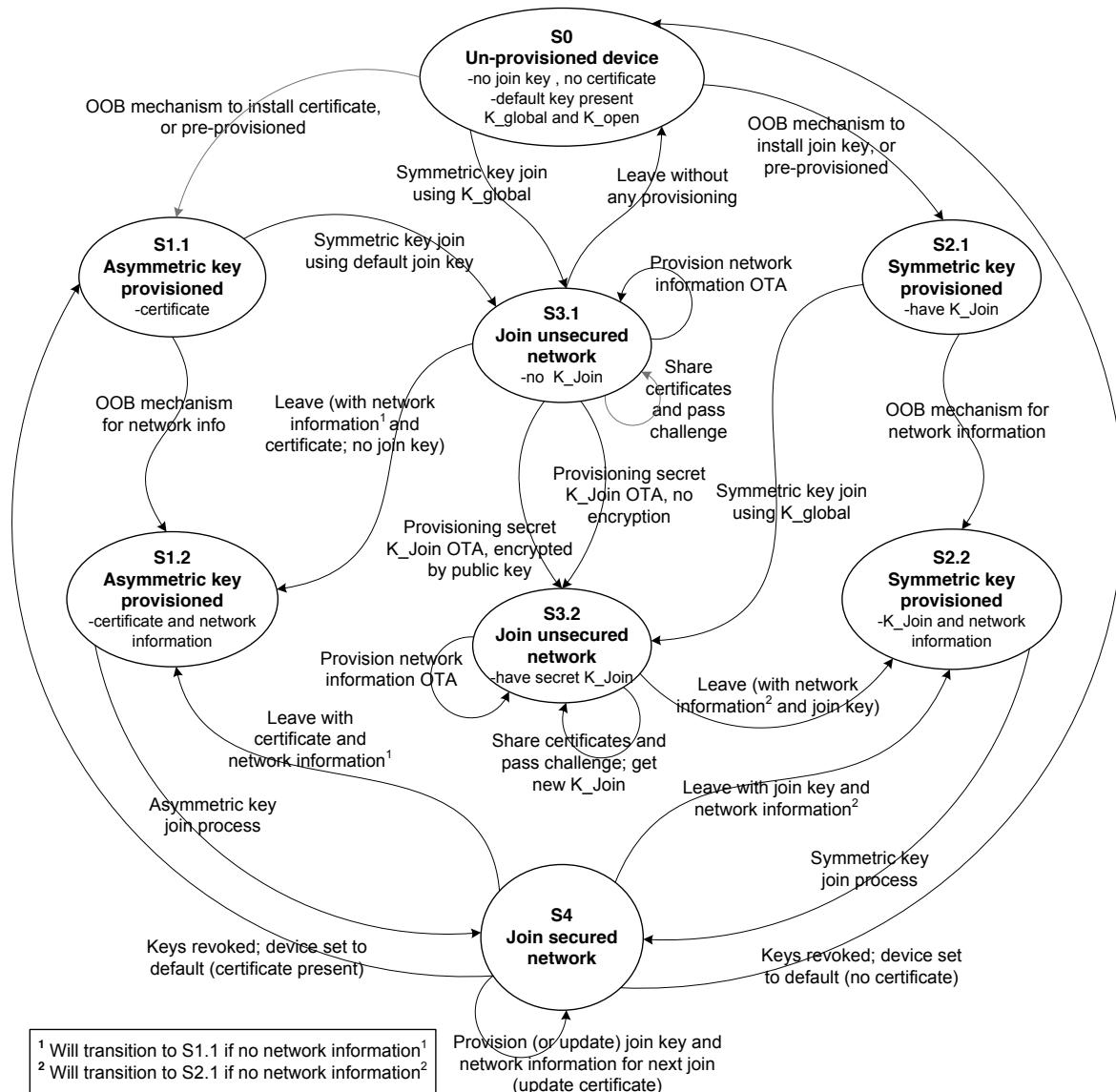
NOTE The interface for such inter-manager communication is beyond the scope of this standard.

---

<sup>10</sup>The two-certificate chain described here is only one of the many certificate topologies possible with multiple certificates. The DPO provides attributes to include multiple certificates.

- 2) The DPSO in the current system manager installs information into the DPO of the device.
- 3) The DBP leaves the current network.
- 4) When the device leaves the current network, it joins the next network with network and security information installed in its DPO.

As described herein, there are multiple paths (and state transitions) available for an un-provisioned device to be provisioned and ultimately to join a secured network. These paths are illustrated via the state transition diagram in Figure 167. Figure 167 is related (and equivalent to) to Figure 166; however, Figure 167 is depicted from the perspective of a device internal state.



**Figure 167 – State transition diagram showing various paths to joining a secured network**

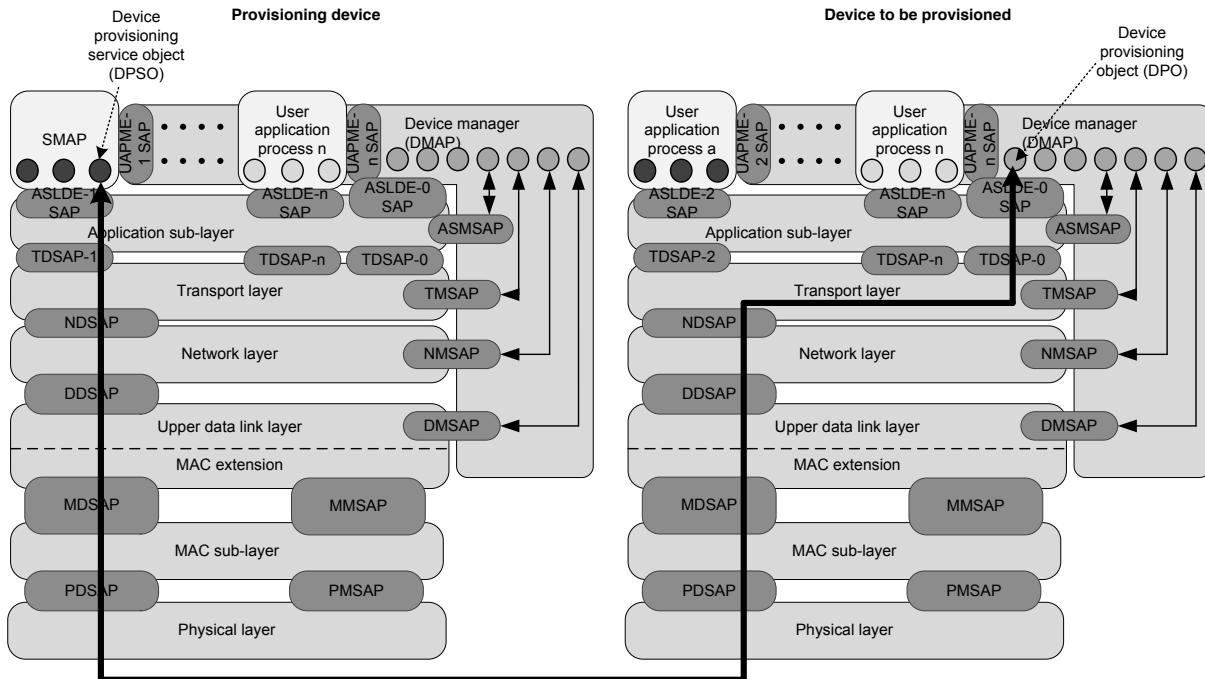
The transitions and paths addressed in Figure 167 include:

1. OOB provisioning of symmetric key and network information:
  1. State transitions : S0 -> S2.1-> S2.2 -> S4
  2. Synopsis: OOB mechanisms are used to provision a device with the target network join key (S0 → S2.1) and network information (S2.1->S2.2). Then, the device uses the symmetric join procedure (S2.2 -> S4) to join the secured network.

2. Factory pre-provisioned (OOB or otherwise): Asymmetric keys and certificates and OOB provisioning of network information.
  1. State transitions : S0 -> S1.1 -> S1.2 -> S4
  2. Synopsis: A device is factory pre-provisioned with asymmetric keys and certificates (S0 -> S1.1). The device has the necessary information to initiate an asymmetric key join procedure. However, it does not have enough network-related information. This information is provisioned using OOB mechanism (S1.1 -> S1.2). Then, the device uses the asymmetric join procedure to join the secured network (S1.2 -> S4).
3. OOB provisioning of symmetric key information and OTA provisioning of network information:
  1. State transitions : S0-> S2.1-> S3.2 -> S2.2 -> S4
  2. Synopsis: A device is provisioned using OOB mechanism (or pre-provisioned) with the AES join key for the target network (S0 ->S2.1). The device then joins a default provisioning network using the default join key, K\_global (S2.1 -> S3.2). The PD in the provisioning network provides the network information for the target network. The device leaves the provisioning network (S3.2 -> S2.2) and joins the secured network (S2.2 -> S4) using the symmetric join procedure.
4. Factory pre-provisioned (OOB or otherwise) asymmetric keys and certificates and OTA provisioning of AES symmetric keys:
  1. State transitions: S0 -> S1.1 -> S3.1-> S3.2 -> S2.2 -> S4
  2. Synopsis: A device is factory pre-provisioned with asymmetric keys and certificates (S0 -> S1.1). The device has the necessary information to initiate an asymmetric key join procedure. However, it cannot join a target network that does not support asymmetric join process. The device then joins a default provisioning network that is different from the target network using the default join key, K\_global (S1.1 -> S3.1). As part of this provisioning network, the device exchanges its credentials, passes a challenge-response mechanism, and receives the target network join key encrypted with the asymmetric key of the device from the PD (S3.1->S3.2). The device is then provisioned with the network information OTA. Then, the device leaves the provisioning network (S3.2 -> S2.2) and joins the secured network (S2.2 -> S4) using the symmetric join procedure.
5. Open join key based provisioning in the clear:
  1. State Transitions : S0 -> S2.1-> S2.2 -> S4(1) -> S2.2 -> S4(2)
  2. Synopsis: A device that has the default open symmetric join key. It uses the symmetric join key procedure for joining a provisioning network (S2.2. -> S4(1)). As part of this provisioning network, the device is provisioned with the target network join key and network information. The device then leaves the provisioning network (S4(1)-> S2.2). The device is now provisioned to join the target network; it joins the secured target network using the symmetric key join process (S2.2 -> S4(2)). In this transition, the first time the device has joined a provisioning network is indicated by state S4(1), and the second time it is joined to the target network is indicated by state S4(2). After the device has reached S4(2), the device cannot use the open symmetric join key unless it is reset to factory defaults.

#### 14.8 Device management application protocol objects for provisioning

This standard defines one device management application process (DMAP) object and one system management application process (SMAP) object for provisioning purposes. The device provisioning object (DPO) holds the configuration settings. Figure 168 illustrates provisioning objects and the interactions between them.



**Figure 168 – Provisioning objects and interactions**

Whether it is the system manager/security manager in a handheld device or the system manager of the target network, the PD shall implement a device provisioning service object (DPSO) with attributes and methods to provision the DBP. The DPSO may have a symmetric key list, used to provision devices that do not have pre-installed keys.

The white list, symmetric key information, and target network information in the DPSO can be maintained with information specific to a device in the White\_List\_Array attribute in Table 425. Alternatively, a pool of valid symmetric keys can be maintained.

When the DBP joins the provisioning network using  $K_{\text{global}}$ , a contract is established between the PD and DBP. The DPSO in the PD can use the established contract to communicate with the DPO in the DBP. Read and write primitives are used for accessing and setting the attributes of the DPO. A subset of network and trust information can now be provisioned in the DPO using the DPSO. To write the new AES join key to the device the DPSO invokes the Write\_Join\_key method of the DPO. This method is allowed if the new key value was received under protection of asymmetric crypto. The attributes in the DPO include both network-related and trust-related information.

NOTE 1 For users requiring additional security, it is recommended that the asymmetric crypto-based authentication and secured key loading technique be used during the trust-related steps while provisioning. Alternatively, OOB mechanisms can be used for provisioning join keys

Once the appropriate trust and network information has been provisioned in the DPO, the device is ready to join the target network. The provisioning network can also be used for device configuration. Since a contract has already been established, the PD may also use the network (or OOB means) to configure the appropriate UAP and DMAP objects of the DBP.

The device provisioning object (DPO) provides an attribute called the Target\_DL\_Config in the DL\_Config\_Info format. DL\_Config\_Info is described in Clause 9 (see Table 103) to configure various attributes of the DL. Once provisioned with this attribute, the DPO provides the DL with an OctetString encapsulating DL\_Config\_Info that includes at least one superframe, and at least one link, that can be used by the DL in searching for advertisements. Target network-specific (e.g., non-default) time slot templates, channel hopping patterns, superframes and links can also be provided to the DBP through the Target\_DL\_Config attribute. Such configuration helps reduce the amount of information (e.g., join superframes) that is otherwise required to be advertised by target network advertisement routers.

The DL of the device plays a major role during the provisioning and joining of the device. The state machine of the DL when it is going through the provisioning process is described in 9.1.14.2.

If the provisioning process is successful, the DPO provides the DL with the set of attributes, including subnet information, superframes, and links, that the DL can use to search for the target network and corresponding subnet(s). In the provisioned state the DL operates its state machine as configured in the superframes and links that were provided by the DPO. Superframe operation may be delayed or disabled by setting the IdleTimer field within the superframe.

Since the device retains the information that was used to provision the DL (all attributes of the DPO), this ensures a means to reset the DL back to its provisioned state by putting the DL into its default state and then adding the provisioned attributes.

The DPO shall be accessible to the system manager of the target network after joining with Key\_Join. Once the device joins a target network, the system manager of the target network has the ability to change the attributes of the DPO. The system manager of the target network has the ability to instruct the device to join another target network by providing network and trust information of the other network. Depending on the value of configuration bit A4, the system manager of the target network has the ability to invoke a DPO.Reset\_To\_Defaults method to remove trust information from the device.

NOTE 2 In the provisioning phase, the DPO in the DBP is accessed by system/security manager functionality in the PD.

## **14.9 Management objects**

### **14.9.1 Device provisioning object**

Table 421 describes the attributes of the DPO. The data type, default value, and a brief description are provided for each attribute. Each attribute also has accessibility of read only or read/write. The attributes of the DPO are accessible only to the system manager/security manager. The value of a read-only attribute can be set only at device manufacturing time (i.e., at a time before the device is certified) or internally by the device; no entity external to the device can change this attribute. Read/write accessibility implies that entities external to the device can change the value of the attribute. The attributes of the DPSO are accessible (read/write) only to the system manager.

The attributes classified as “constant” have a value that is not changed during the device lifecycle, neither internally nor externally. The definition of the classification is found in 12.6.3.

**Table 421 – Device provisioning object**

<b>Standard object type name: DPO (Device provisioning object)</b>				
<b>Standard object type identifier: 120</b>				
<b>Defining organization: ISA</b>				
Attribute name	Attribute identifier	Attribute description	Attribute data information	Description of behavior of attribute
Default_Nwk_ID	1	A published network identification for the default network	Type: Unsigned16	This is the network identification for the default network. The default network may be used to form the provisioning mini-network
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: 1	
			Valid value set: N/A	
Default_SYM_Join_Key	2	A published join key for the default network	Type: Unsigned128	This key is used by devices to join the default network. The default keys may be used to form the provisioning mini-network
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: ISA(space)100(null)	
			Valid value set: N/A	
Open_SYM_Join_Key	3	A published join key for the default network	Type: Unsigned128	This key is used by devices to join the unsecured provisioning network Actual value of this key is defined in 3.3.1.4.
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: OPEN(null)(null)(null)(null)	
			Valid value set: N/A	
Default_Frequency_List	4	The list of frequencies used by the default network. The attribute is coded as a bit map of 16 bits representing the 16 frequencies	Type: Unsigned16	The list of frequencies used by the advertising routers of the default network. To join the default network the device may receive advertisements on any of these frequencies
			Classification: Constant	
			Accessibility: Read only	
			Initial default value: 7FFF	
			Valid value set: N/A	
Join_Method_Capability	5	An enumeration set to indicate the join capabilities of the device.	Type: Array of Boolean	This attribute defines the capability of a device to join. The device can either use symmetric keys or asymmetric key infrastructure to join a target network. This attribute merely defines the capabilities of the device. The actual method used to join the target network shall be set by the PD
			Classification: Constant	
			Accessibility: Read/write	
			Initial default value: 00	
			Valid value set: 00= default join only 01= SYM key join only 10= PKI join only 11= PKI or SYM Key join	
Allow_Provisioning	6	A Boolean value set to indicate if a device is allowed to be provisioned or not	Type: Boolean	This flag is used to lock the state of an already provisioned device. If this value is set the device will not accept any reads or writes to the target network attributes
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Allowed (1)	
			Valid value set: 0= not allowed to be provisioned 1= Allowed to be provisioned	
Allow_Over_The_Air_Provisioning	7	A Boolean value set to indicate if a device is allowed to be provisioned or not	Type: Boolean	This flag indicates if over the air provisioning is enabled or disabled in this device. If over the air provisioning is disabled the
			Classification: Static	
			Accessibility: Read/write	
			Initial default value: Allowed (1)	
			Valid value set: 0= only out	

<b>Standard object type name: DPO (Device provisioning object)</b>				
<b>Standard object type identifier: 120</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
			of band provisioning 1= Allowed to be provisioned over air	device needs to be provisioned using out of band methods. Backbone devices shall have this value set to (0); In either case, provisioning is allowed only if the Allow_Provisioning attribute is enabled
Allow_OOB_Provisioning	8	A Boolean value set to indicate if a device is allowed to be provisioned using OOB means	Type: Boolean Classification: Static Accessibility: Read/write Initial default value: Allowed (1) Valid value set: 0= not allowed 1= allowed	The flag is used to block the devices from accepting provisioning information from OOB means
Allow_Reset_to_Factory_defaults	9	A Boolean value set to indicate if a device is allowed to be reset to factory defaults	Type: Boolean Classification: Static Accessibility: Read/write Initial default value: Allowed (1) Valid value set: 0= not allowed 1= allowed	This flag is used to block devices from being reset to factory defaults by a system manager.
Allow_Default_Join	10	A Boolean value set to indicate if a device is allowed to join a network using the default keys	Type: Boolean Classification: Static Accessibility: Read/write Initial default value: Allowed (1) Valid value set: 0= not allowed 1= allowed	The flag is used to force the devices to join a particular target network and not join to any default network. Devices choosing not to join a Default network can set this attribute to 0
Target_Nwk_ID	11	The network ID of the target network that this device is provisioned to join	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: N/A	This attribute indicates the target network that this device has to join. If the Target_Nwk_BitMask (attribute 12) is set to 0xFFFF (all 0x1), the device shall ignore advertisements from routers belonging to any other network except the indicated target network. Otherwise a combination of network ID and bit mask shall be used. (See description of attribute 12 on how the NetworkID and bitmask are combined). This helps with fast joins and also prevents devices from trying to join all networks in its vicinity. This value can be set to 0

<b>Standard object type name: DPO (Device provisioning object)</b>						
<b>Standard object type identifier: 120</b>						
<b>Defining organization: ISA</b>						
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>		
Target_Nwk_BitMask	12	A bit mask for matching of the bits of the Target network ID	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 1111111111111111 Valid value set= [0 – xFFFF]	to allow responses to any advertising router		
Target_Join_Method	13	A Boolean value to indicate if the device should use SYM key join or Asymmetric Key join mechanism to join the target network.	Type: Boolean Classification: Static Accessibility: Read/write Initial default value: 1 Valid value set: 0= Symmetric key 1= Asymmetric key	The bit mask is useful for matching multiple target networks. If the value of a bit in the bit mask is 1 then the bit has to be exactly matched to the corresponding bit in the Target Network. The default value of all 1s indicates that all bits of network ID need to match		
Target_Security_Manager_EUI	14	The EUI-64 address of the security manager in the target network that the device is intended to join	Type: Unsigned64 Classification: Static Accessibility: Read/write Initial default value: 0xFF.FF (all 0xFF) Valid value set: An extended 64 bit IEEE address	Subclause 7.4.4 defines two different methods for join depending on the use of either symmetric keys or PKI. This attribute sets method to be used to join a target network.		
Target_System_Manager_Address	15	The 128-bit network address of the system/security manager in the target network that the device is intended to join	Type: Unsigned128 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: An extended 64 bit IEEE address	Set to the EUI-64 address of the security manager that the device is provisioned to join		
Target_Frequency_List	16	The list of frequencies used by the target network. The attribute is coded as a bit map of 16 bits representing the 16 frequencies	Type: Array of Unsigned16 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A	The 128 bit network address is required for backbone devices to join the network. The backbone devices do not have an advertising router - hence a join request is sent to the 128-bit network address of the system/security manager to begin the join process. I/O devices and routing devices need not be provisioned with this attribute		

<b>Standard object type name: DPO (Device provisioning object)</b>				
<b>Standard object type identifier: 120</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
Target_DL_Config	17	The data link layer (DL) configuration information for this device.	Type: OctetString Classification: Static Accessibility: Read / Write Initial default value: N/A Valid value set: N/A	This attribute indicates the various configuration settings for the data link layer of the device. The structure of this attribute is defined in the DL_Config_Info defined in Clause 9
PKI_Certificate_Type	18	(optional) The type of certificate stores in PKI_Root_Certificate and PKI_Certificates	Type: Unsigned8 Classification: Static Accessibility: Read/Write Initial default value: N/A Valid value set: 0 (Implicit certificate)  0 = Implicit certificate 1 = manual certificate	This field indicates a type of Certificate in PKI_Root_Certificate and PKI_Certificates.  0 = Implicit certificate 1 = manual certificate
PKI_Root_Certificate	19	(optional)The root certificate of the Certificate Authority issuing the certificate to the device.	Type: OctetString Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A	The root certificate of the certificate authority and its corresponding asymmetric key is used to verify certificates of the peer nodes. The root certificate may updated by the system manager
Number of PKI_Certificates	20	(optional)The number of certificates stored in the PKI_certificate attribute	Type: Unsigned8 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: N/A	This field indicates the number of certificates available in attribute 20
PKI_Certificate	21	(optional)The certificate issued to this device for joining using the asymmetric key infrastructure.	Type: Array of OctetStrings Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A	If Target_Join_Method is set to PKI, this attribute contains the certificate (which includes the asymmetric key, device ID, and other text) signed by a certificate authority which is required for joining the target network
Current_UTC_Adjustment	22	The current value of the UTC accumulated leap second adjustment	Type: Integer16 Classification: Static Accessibility: Read/write Initial default value: 34 Valid value set: -32768 to + 32767	Devices that need to convert TAI time to hh:mm:ss format need this adjustment; units in seconds; note that the adjustment can be negative; note that UTC and TAI are based on different start dates but this difference is not covered by this attribute; on January 1 2009 the value changed from 33 sec to 34 sec

#### 14.9.2 Device provisioning object methods and alerts

Several methods and alerts are available in the DPO. Table 422 describes the Reset\_To\_Default method.

**Table 422 – Reset\_To\_Default method**

<b>Standard object type name(s): Device provisioning object</b>			
<b>Standard object type identifier: 120</b>			
<b>Defining organization: ISA</b>			
<b>Method name</b>	<b>Method ID</b>	<b>Method description :</b>	
Reset_To_Default	1	This method is used to reset to default settings for the provisioning. This method shall be executed only when Allow_Provisioning is enabled.	
<b>Input arguments (None)</b>			
<b>Output arguments</b>			
	<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>
	1	Status	Unsigned8
			0= success, 1= failure,

Table 423 describes the method to write a symmetric join key. The join key shall not be exposed to a remote device, and may be exposed to limited internal process; DPO.Write\_Symmetric\_Join\_Key() method installs a join key to a memory area that is not used for attributes (e.g., secure storage).

**Table 423 – Write symmetric join key method**

<b>Standard object type name(s): Device provisioning object</b>			
<b>Standard object type identifier: 120</b>			
<b>Defining organization: ISA</b>			
<b>Method name</b>	<b>Method ID</b>	<b>Method description :</b>	
Write_SYM_join_key	2	This method is used to write a symmetric AES join key to a device. This method is evoked by the DPSO to provision a DBP with the Target AES join key. Depending on the provisioning method used this method call APDU and hence the join key may be encrypted by the session key between the device and PD alone or the device's asymmetric key in the APDU in addition to the APDU being encrypted by session key. This method shall be executed only when Allow_Provisioning is enabled	
<b>Input arguments</b>			
	<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>
	1	New_Key_Value	Unsigned128
	2	Encrypted By	Boolean
			0= AES_TL_Session_Key_Only, 1= ASYM_Asymmetric_Key
<b>Output arguments</b>			
	<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>
	1	Status	Unsigned8
			0= success, 1= failure,

#### 14.10 Device provisioning service object

##### 14.10.1 Device provisioning service object attributes

Table 424 describes the attributes of the DPSO.

The system manager can either choose particular provisioning information for each EUI-64 or a set of join keys for a set of EUI-64s with no one-to-one mapping. The Boolean attribute, DPSO.Enable\_White\_List\_Array, is used to specify which method is used.

In the DPSO.Enable\_White\_List\_Array set, DPSO.White\_List\_Array is used to install particular provisioning information for each EUI-64 of the DBP.

If DPSO.Enable\_White\_List\_Array is not set, the PD shall check if there are at least as many entries in DPSO.SYM\_Key\_List as entries in DPSO.White\_List. This specification does not specify how each entry in DPSO.SYM\_Key\_List and DPSO.White\_List is mapped.

**Table 424 – Device provisioning service object**

<b>Standard object type name: DPSO (Device provisioning service object)</b>				
<b>Standard object type identifier: 106</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
White_List	1	A list of devices permitted to be provisioned by this object	Type: Array of Unsigned64 Classification: Static Accessibility: Read/write Initial default value: [] Valid value set: N/A	This list contains EUI-64 address of device being provisioned. This list can be used to restrict a provisioning device to provision only a set of devices whose EUI-64 address in this list. If this attribute is NULL, then the provisioning device can provision any device
SYM_Key_List	2	A list of valid join keys with which a device can be provisioned.	Type: Array of Unsigned128 Classification: Static Accessibility: Read/write Initial default value: [ISA100] Valid value set: N/A	This key is used by devices to join the target network, that have suitable entropy
SYM_Key_Expiry_Times	3	The expiration time for each key (optional)	Type: Array of TAITimeRounded Classification: Static Accessibility: Read/write Initial default value: FFFFFFFF (Infinity) Valid value set: N/A	This attribute sets the expiry time for each of the symmetric keys. The key is only used for provisioning if it has not expired.
Target_NWK_ID	4	The network ID of the target network that the devices provisioned by this object are supposed to join	Type: Unsigned16 Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: N/A	This attribute indicates the target network (subnet ID) that a provisioned device has to join
Target_Join_Method	5	A Boolean value to indicate if the devices provisioned by this object should use SYM key join or Asymmetric Key join mechanism to join the target network	Type: Boolean Classification: Static Accessibility: Read/write Initial default value: 0 Valid value set: 0= Symmetric key 1= Asymmetric Key	Clause 7 defines two different methods for join depending on the use of either symmetric keys or PKI. This attribute sets method to be used to join a target network
Target_Security_Manager_EUI	6	The EUI-64 address of the Security Manager in the target network that the device provisioned by this object is intended to join	Type: Unsigned64 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: Valid IEEE EUI64 address	Set to the EUI-64 address of the security manager that the device is provisioned to join
Target_System_Manager_Address	7	The 128-bit network address of the System/Security	Type: Unsigned128 Classification: Static Accessibility: Read/write	The 128 bit network address is required for backbone devices to join the network

<b>Standard object type name: DPSO (Device provisioning service object)</b>				
<b>Standard object type identifier: 106</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
		Manager in the target network that the device provisioned by this object is intended to join	Initial default value: N/A Valid value set: 0 – 2 <sup>128</sup> -1	
Target_Frequency_List	8	The list of frequencies used by the default network. The attribute is coded as a bit map of 16 bits representing the 16 frequencies	Type: Array of Unsigned16 Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A	The target network may be using only a subset of frequencies for advertisements by the join routers
Target_DL_Config	9	The data link layer (DL) configuration information for the device to be provisioned by this object	Type: OctetString Classification: Static Accessibility: Read / Write Initial default value: N/A Valid value set: N/A	This attribute indicates the various configuration settings for the data link layer of the device. The structure of this attribute is defined in the DL_Config_Info defined in Clause 9, Table 103
Allow_Provisioning	10	A Boolean value set to indicate if a device is allowed to be provisioned again or not	Type: Boolean Classification: Static Accessibility: Read/write Initial default value: Allowed (1) Valid value set: 0= not allowed to be provisioned 1= Allowed to be provisioned	This flag is used to lock the future state of a provisioned device
Allow_Default_Join	11	A Boolean value set to indicate if a device provisioned by this object is allowed to join a network using the default keys	Type: Boolean Classification: Static Accessibility: Read/write Initial default value: Not_allowed (0) Valid value set: 0= not allowed 1= allowed	The flag is used to force the provisioned devices to join a particular target network and not join to a default network. Once provisioned the device should join the target network
Enable_White_List_Array	12	A Boolean value set to indicate if the provisioning object is designed to set device specific provisioning information	Type: Boolean Classification: Constant Accessibility: Read/write Initial default value: 0=disabled Valid value set: 0= disabled 1=enabled	IF this flag is set the DPSO is capable of provisioning different devices (based on the EUI-64 address) with different provisioning information. This can be used to provision a particular device, with a particular security manager and a particular Target_Join_Time, for example

<b>Standard object type name: DPSO (Device provisioning service object)</b>				
<b>Standard object type identifier: 106</b>				
<b>Defining organization: ISA</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
White_List_Array	13	(Optional) An array of the EUI addresses that the DPSO intends to provision along with the corresponding provisioning information for that device	Type: Array of DPSOWhiteListTbl structures Classification: Static Accessibility: Read/write Initial default value: N/A Valid value set: N/A	This attribute shall be used only if Device_specific_provisioning_flag is set. It contains the device specific provisioning information like device specific join keys, device specific target security manager etc.
White_List_Array_Meta	14	Metadata for White List Array (Attribute 13) or set of White_List (Attribute 1) and SYM_Key_List (Attribute 2)	Type: Meta_Data_Attribute Classification: Static Accessibility: Read only Initial default value: N/A Valid value set: N/A	Metadata containing a count of the number of entries and capacity (the total number of rows allowed) of White_List_Array table or set of White_List . If Enable_White_List_Array is enabled, this attribute specifies count of the number of entries and capacity for White_List_Array. Enable_White_List_Array is disabled, this attribute specifies count of the number of entries and capacity for set of White_List and SYM_Key_List.
DPSO_Alerts_AlertDescriptor	15	Used to change the priority of DPSO alerts that belong to the security category; these events can also be turned on or turned off	Type : Alert report descriptor Classification: Static Accessibility: Read/write Initial default value: Alert report disabled = False Alert report priority = 6 Valid value set: See type definition	See description of alerts in Table 427 and Table 428
Current.UTC.Adjustment	16	The current value of the UTC accumulated leap second adjustment	Type: Integer16 Classification: Static Accessibility: Read/write Initial default value: 34 Valid value set: - 32768 to + 32767	Devices that need to convert TAI time to hh:mm:ss format need this adjustment; units in seconds; note that the adjustment can be negative; note that UTC and TAI are based on different start dates but this difference is not covered by this attribute; on January 1 2009 the value changed from 33 s to 34 s

#### 14.10.2 Device provisioning service object structured attributes

Table 425 describes the structured attributes of the DPSO. The structured attribute DPSO.DPSO.White\_List\_Array is used when the PD has device-specific information, i.e., if the PD has symmetric join keys and other DPO attributes that are specific to a device. In this case, a structured array is required that stores the provisioning information indexed by the EUI-64 identifier of the DBP. This indexed array is described in Table 425.

After the PD receives the Device\_SYM\_Key from the security manager, the PD shall not expose the Device\_SYM\_Key attribute externally.

NOTE The interface between the PD and the security manager is out of scope of this standard.

**Table 425 – DPSOWhiteListTbl data structure**

<b>Standard data type name: DPSOWhiteListTbl</b>		
<b>Standard data type code: 440</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
Device_EUI	1	Type: Array of Unsigned64
		Classification: Constant
		Accessibility: Read only
		Initial default value: 1
		Valid value set: N/A
Device_Tag	2	Type: OctetString
		Classification: Constant
		Accessibility: Read/write
		Initial default value: Device-EUI as OctetString
		Valid value set: N/A
Device_SYM_Key	3	Type: Array of Unsigned128
		Classification: Constant
		Accessibility: Read only
		Initial default value: ISA100
		Type: OctetString
SYM_Key_Expiry_Time	4	Type: Array of TAI Unsigned40
		Classification: Static
		Accessibility: Read only
		Initial default value: FFFFF (Infinity)
		Valid value set: N/A
Target_Nwk_ID	5	Type: Unsigned16
		Classification: Static
		Accessibility: Read only
		Initial default value: 0
		Valid value set: N/A
Target_Join_Method	6	Type: Boolean
		Classification: Constant
		Accessibility: Read only
		Initial default value: 1
		Valid value set: 0= symmetric key 1= asymmetric key
Target_Security_Manager_EUI	7	Type: Unsigned64
		Classification: Static
		Accessibility: Read only
		Initial default value: N/A
		Valid value set: 0 – 2 <sup>64</sup> -1
Target_System_Manager_Address	8	Type: Unsigned128
		Classification: Constant
		Accessibility: Read only
		Initial default value: N/A
		Valid value set: 0 – 2 <sup>128</sup> -1
Target_Frequency_List	9	Type: Array of Unsigned8
		Classification: Constant
		Accessibility: Read only
		Initial default value: N/A
		Valid value set: N/A
Target_DL_Config	10	Type: OctetString (See DL_Config_Info for format)
		Classification: Static
		Accessibility: Read only
		Initial default value: N/A
		Valid value set: N/A
Allow_Provisioning	11	Type: Boolean
		Classification: Static

<b>Standard data type name: DPSOWhiteListTbl</b>		
<b>Standard data type code: 440</b>		
<b>Element name</b>	<b>Element identifier</b>	<b>Element scalar type</b>
		Accessibility: Read/write
		Initial default value: 1
		Valid value set: 0= not allowed to be provisioned 1= Allowed to be provisioned
Allow_Default_Join	12	Type: Boolean Classification: Static Accessibility: Read/write Initial default value: 1 Valid value set: 0= not allowed 1= allowed

When set to a value other than the OctetString representation of the EUI-64, Device\_Tag represents a Tag name assigned to the device by a user. This value shall be written to the Tag\_Name attribute (attribute #7) of the DMO (see 6.2.8).

#### 14.10.3 Device provisioning service object methods

Several methods are available for manipulating the DPSO. Standard methods such as read and write can be used for scalar or structured MIBs (SMIBs) in their entirety. The methods described herein are used to manipulate tables. The methods in this subclause allow access to a particular row of an SMIB based on a unique key field.

It is assumed that the tables have a unique key field, which may either be a single element or the concatenation of multiple elements. The key field is assumed to be the (concatenation of) the first (few) element(s) of the table.

Table 426 describes the methods for manipulation of structured MIBs. These methods are based on the Read\_Row, Write\_Row and Delete\_Row templates defined in Annex J.

**Table 426 – Array manipulation table**

<b>Standard object type name(s): Device provisioning service object</b>		
<b>Standard object type identifier: 106</b>		
<b>Defining organization: ISA</b>		
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>
Setrow_WhiteListTbl	1	Method to set (either write or edit) the value of a single row of the white list array. The method uses the Write_Row method template defined in Annex J with the following arguments: Attribute_ID :14 (White_List_Array) Index 1: 1 (Device_EUI)
Getrow_WhiteListTbl	2	Method to get the value of a single row of the white list array. The method uses the Read_Row method template defined in Annex J with the following arguments: Attribute_ID :14 (White_List_Array) Index 1: 1 (Device_EUI)
Deleterow_WhiteListTbl	3	Method to delete the value of a single row of the white list array. The method uses the Delete_Row method template defined in Annex J with the following arguments: Attribute_ID :14 (White_List_Array) Index 1: 1 (Device_EUI)

#### 14.10.4 Device provisioning service object alerts

Table 427 describes an alert to indicate a join attempt by a device that is not on the white list.

**Table 427 – DPSO alert to indicate join by a device not on the WhiteList**

<b>Standard object type name(s): Device provisioning service object</b>					
<b>Standard object type identifier: 106</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Alert to indicate provisioning request by a device not on white list</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority (Enumerated: high, med, low, journal only)	Value data type	Description of value included with alert
0 = Event	2 = Security	0 = Not_On_Whitelist_Alert	6 = Medium	Type: Unsigned64	EUI-64 of the device not on white list

Table 428 describes an alert to indicate that inadequate capability is available for a device to join the network.

**Table 428 – DPSO alert to indicate inadequate device join capability**

<b>Standard object type name(s): Device provisioning service object</b>					
<b>Standard object type identifier: 106</b>					
<b>Defining organization: ISA</b>					
<b>Description of the alert: Alert to indicate inadequate device join capability</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority (Enumerated: high, med, low, journal only)	Value data type	Description of value included with alert
0 = Event	2 = Security	1 = Inadequate_Join_ Capability_Alert	6 = Medium	Type: OctetString	The Value is an octet string consisting of at least 9 octets  The first octet is an enumeration of the diagnostic code with the following values: 1 – Bad join key 2 – Expired join key 3 – Authentication failed  The next 8 octets specify the EUI-64 of the device that tried to join

#### 14.10.5 Summary of attributes that can be provisioned

The following is a summary of the attributes that are provisioned by the PD so that a new device can join a target network. These attributes can be provisioned either using over the air (OTA) methods or OOB methods. The list of provisioned attributes includes the follow items. The full list is defined by Table 421, Table 424, and Table 425.

- Trust-related information :
- Join key (K\_join)
- EUI-64 of security manager
- Network join method
- Network-related information:
- Network ID and bitmask
- 128-bit address of system manager

- DL config (contains the superframes, link TsTemplate, channel information, etc., needed to join the target network)

In addition, the configuration bits (attributes 6-10 of the DPO), describing the behavior of the device, can be set by the provisioning device.

## **14.11 Provisioning functions (INFORMATIVE)**

### **14.11.1 General**

The provisioning interface and procedures described herein do not describe a human-machine interface (HMI). This standard does not specify a specific HMI, but does describe how such tools can be designed for provisioning devices conforming to this standard. In plant operations, a user may enter provisioning data and accept or reject devices wishing to be provisioned, using a handheld device or an interface at some central location.

Provisioning scenarios in this clause are examples of provisioning using methods described by the specification.

### **14.11.2 Examples of provisioning methods**

#### **14.11.2.1 General**

This subclause describes some examples of how the management objects and procedures described can be used to provision a device. The examples herein discuss the following provisioning methods:

- Provisioning over-the-air using pre-installed join keys.
- Provisioning using out of band mechanisms.
- Provisioning over-the-air using PKI certificates.
- Provisioning over-the-air using dual role advertisement routers.
- Provisioning backbone devices.

#### **14.11.2.2 Provisioning over-the-air using pre-installed join keys**

The steps for provisioning a device with pre-installed trust information may include:

- 1) The device arrives at the deployment site with pre-installed keys. The keys are symmetric join keys.
- 2) A WhiteList of device addresses and their corresponding symmetric keys is installed in the security manager of the target network. The mechanism by which these keys are installed in the security manager is beyond the scope of this standard. The WhiteList and corresponding symmetric keys may be securely emailed, sent on CDs, hand delivered and keyboard entered, or delivered using any other appropriate tool.
- 3) The target network may be using a subset of frequencies allowed and may be operating in the vicinity (i.e., in interference range) of multiple distinct networks conforming to this standard. In this case, network-related information may be provisioned in the device. This information allows the device to respond to advertisements from target networks only, and also to listen for advertisements at the correct frequencies and at the correct time, thereby decreasing interference and decreasing join times. The network-related information is provisioned using a PD via an out-of-band communication mechanism or via over-the-air mechanisms. This step is optional. If not provisioned with specific target network information, the device may try all frequencies and attempt to join all networks in its vicinity.
- 4) The device listens for advertisements from an advertising router in the target network. Once an advertisement is heard, the device sends a join request to the system manager/security manager of the target network. (The join process is described in 7.4.)
- 5) The system/security manager checks its WhiteList and then checks to see if the join key of the device matches the join key for the device provided to the security manager. If the join keys match, the security manager provides a master key to the device that is a shared secret key between the security manager and the device. In addition, session keys and DL

keys are provided, and a contract is established with the new device to complete the join process.

#### **14.11.2.3 Provisioning using out-of-band mechanisms**

The steps for provisioning a device using an OOB provisioning device might include:

- 1) A fresh device arrives at the user site. The device has default settings and no-pre-installed keys.
- 2) A PD (e.g., a handheld) obtains a list of symmetric keys generated by the security manager/ system manager of the target network. The symmetric keys are time-bounded. The keys may be an array of keys or device-specific key/EUI-64 pairs at the discretion of the security manager/system manager. This information is stored in the PD.
- 3) The handheld device, loaded with the symmetric keys, is brought near the new device or connected to the new device and an OOB connection is made between the handheld device and the DBP. The handheld device then uses the OOB communication interface to populate the attributes of the DPO in the new device. OOB communication may occur over infrared, physical connection, near field communication, or other means.
- 4) The device is now ready to join the target network and listens for advertisements from the target network. The device responds to the advertisements by sending a join request through the advertising router to the target system manager. The security manager checks its WhiteList, if applicable. The security manager also checks the validity of the join key and verifies that the key has not expired. If the key is valid, the security manager accepts the join request and provides the device with a master key that is a shared secret key between the security manager and the device. In addition, session keys and DL keys are also provided and a contract is established with the new device to complete the join process.

#### **14.11.2.4 Provisioning over-the-air using asymmetric key infrastructure certificates**

The steps for provisioning a device that has pre-installed trust information might include:

- 1) The device arrives at the deployment site with an installed security module. The module contains a factory-signed certificate and a public/private key pair. A certificate authority (CA) has signed the issuer key of the factory.
- 2) A WhiteList of device addresses and a list of asymmetric keys of certificate authorities are installed in the security manager of the target network. The mechanism by which these keys are installed in the security manager is beyond the scope of this standard. The WhiteList may be securely emailed, sent on CDs, hand delivered and entered via keyboard, or delivered using any other appropriate tool.
- 3) (optional) The target network may be using a subset of frequencies allowed; it may be operating in the vicinity (i.e., within interference range) of multiple networks. In such a case, network-related information may be provisioned into the device. This information allows the device to respond to advertisements only from target networks, and to listen for advertisements on the correct frequencies at the correct time, thereby decreasing interference and increasing join times. If the device is not provisioned with specific target network information, the device may try all frequencies and may try to join all networks in its vicinity. The network-related information is provisioned using a provisioning device via an OOB communication mechanism or via over-the-air mechanisms.
- 4) The device now listens for the advertisements from the advertising router in the target network. Once an advertisement is heard, the device shares its certificates with the security manager. With the CA asymmetric key, the security manager decodes the certificates and checks that they are valid. The procedure also involves a challenge-response mechanism on part of the system manager/security manager to confirm the identity of the joining device. The security manager checks its WhiteList to confirm that the device is intended to join the network. The confirmation step may involve a pop-up on a GUI of the security manager for manual confirmation by a user. Once confirmed, the security manager may issue session keys for the device if the device wishes to join the network immediately. Optionally, the security manager may issue symmetric join keys for the device to join the network at a later time. In either case, the issued keys are sent back to the device, encrypted with the asymmetric key of the device.

#### **14.11.2.5 Provisioning over-the-air using dual role advertisement routers**

The steps for provisioning a device over-the-air using a dual role advertisement router might include:

- 1) The device arrives with factory default settings at a user site. The user site requires very low levels of security.
- 2) Some of the advertisement routers at the user site have a dual role and function as provisioning devices. Using the open symmetric join key ( $K_{join} = K_{open}$ ), the dual role advertisement router (i.e., the logical PD side of the dual role advertisement router) forms a mini-network with the new device and provides the new device with the network settings and join key for the target network. These settings, including the keys, are sent in the clear over the air. The dual role device may also inform the security manager of the target network to update its white list by adding the device that has just been provisioned. The dual role provisioning device may be operational in a place where the user is fairly confident that transmission of join keys over the air poses little risk. (This step poses similar risk as that in binding garage door openers to remote controls.)
- 3) The DPO of the new device now has the trust information and network information to join the target network. It can use the advertisement routers (either the same dual role advertisement router that provisioned it, or some other advertisement router of the target network if the device was moved) and sends a join request to the system manager of the target network.
- 4) The system manager of the target network accepts the join request and provides a contract to the new device.

#### **14.11.2.6 Provisioning backbone devices**

The steps for provisioning backbone devices might include:

- 1) A fresh device arrives at the user site. The device has default settings and no pre-installed keys.
- 2) A PD (e.g., a handheld or a device connected via the backbone interface to the DBP) obtains a list of symmetric keys generated by the security manager/ system manager of the target network. The symmetric keys are time-bounded. The keys may be an array of keys or device-specific key/EUI-64 pairs at the discretion of the security manager/system manager. This information is stored in the DPSO of the PD.
- 3) The PD loaded with the symmetric keys is brought near the new device or connected to the new device and an OOB connection (which can, in this case, be the backbone) is made between the handheld device and the DBP. The PD then uses the OOB communication interface (most probably the backbone interface) to populate the attributes of the DPO in the new device.
- 4) The device is now ready to join the target network. However, unlike a field device, a backbone device may not have a DL interface. For example, the device may be a gateway residing on the backbone. Alternatively, the backbone device may be the first advertising router connected to the network. For example, the device may be a backbone router with advertisement router functionality on the IEEE Std 802.15.4 physical layer interface. However, the device needs to be provisioned over the backbone and not through the PhL, since in this case there are no advertising routers that can forward its join request to the system manager.

To talk to the system manager on the backbone without the help of an advertising router, the backbone router sends a join request to the system manager directly over the backbone; the backbone device can form the network header necessary to send this message. It can do so because it has been provisioned with the 128-bit address of the system manager (DPO.Target\_System\_Manager\_Address). The remaining procedure at the system manager is same as that in 14.11.2.3.

## **Annex A (INFORMATIVE)**

### **Protocol implementation conformance statement proforma**

#### **A.1 Introduction**

##### **A.1.1 General**

To evaluate conformance of a particular implementation, it is necessary to have a statement of those capabilities and options that have been implemented for a given standard. Such a statement is called a protocol implementation conformance statement (PICS). The PICS collects mandatory and optional behaviors from the body of the standard and summarizes them with references to the text where the behavior is stated. Since the PICS does not create any normative text it has been labeled as informative.

##### **A.1.2 Scope**

This annex provides the PICS proforma for this standard in compliance with the relevant requirements, and in accordance with the relevant guidance as given in ISO/IEC 9646-7:1995.

##### **A.1.3 Purpose**

The supplier of a protocol implementation claiming to conform to this standard shall complete the following PICS proforma and accompany it with the information necessary to identify fully both the supplier and the implementation.

The PICS of a protocol implementation is a statement of which capabilities and options of the protocol have been implemented. The statement is in the form of answers to a set of questions in the PICS proforma. The questions in this proforma consist of a systematic list of protocol capabilities and options as well as their implementation requirements organized by protocol stack layer. The implementation requirement indicates whether implementation of a capability is mandatory, optional, or conditional depending on options selected. When a protocol implementer answers questions in a PICS proforma, the implementer indicates whether an item is implemented and provides explanations if an item is not implemented.

##### **A.1.4 Abbreviations and special symbols**

Abbreviations and symbols used include:

- Notations for requirement status:
- M-Mandatory
- O- Optional
- O.n - Optional, but support of at least one of the group of options labeled O.n is required.
- N/A- Not applicable
- X - Prohibited

For example, FD1: O.1 and FD2: O.1 indicate that the status is optional but at least one of the features described in FD1 and FD2 is required.

PICS list that behavior that is mandated by the standard. It also includes that behavior that is mandatory when an optional behavior is cited

##### **A.1.5 Instructions for completing the PICS proforma**

If it is claimed to conform to this standard, the actual PICS proforma to be filled in by a supplier shall be technically equivalent to the text of the PICS proforma in this annex and shall preserve the numbering, naming, and ordering of the PICS proforma.

The main part of the PICS is a fixed-format questionnaire, divided into five tables. Answers to the questionnaire are to be provided in the rightmost column, either by simply marking an answer to indicate a restricted choice (such as yes or no) or by entering a value or a set or range of values.

**A.1.6 Identification of the implementation****A.1.6.1 Implementation under test identification**

Implementation under test (IUT) name:

---

IUT model number and version: \_\_\_\_\_  
\_\_\_\_\_

---

**A.1.6.2 System under test identification**

System under test (SUT) name:

---

Hardware configuration: \_\_\_\_\_  
\_\_\_\_\_

---

Operating system: \_\_\_\_\_  
\_\_\_\_\_

---

**A.1.6.3 Application layer identification**

AL vendor name:

---

**A.1.6.4 Product supplier**

Name: \_\_\_\_\_

Address: \_\_\_\_\_  
\_\_\_\_\_

---

Telephone number: \_\_\_\_\_

Facsimile number: \_\_\_\_\_

Email address: \_\_\_\_\_

Additional information: \_\_\_\_\_

**A.1.6.5 Client**

Name: \_\_\_\_\_

Address: \_\_\_\_\_  
\_\_\_\_\_

---

Telephone number: \_\_\_\_\_

Facsimile number: \_\_\_\_\_

Email address: \_\_\_\_\_

Additional information: \_\_\_\_\_

**A.1.6.6 Protocol implementation conformance statement contact person**

Name: \_\_\_\_\_

Address: \_\_\_\_\_  
 \_\_\_\_\_

Telephone number: \_\_\_\_\_

Facsimile number: \_\_\_\_\_

Email address: \_\_\_\_\_

Additional information: \_\_\_\_\_

#### **A.1.6.7 Protocol implementation conformance statement /system conformance statement**

Provide the relationship of the PICS with the system conformance statement for the system:

\_\_\_\_\_  
 \_\_\_\_\_  
 \_\_\_\_\_

#### **A.1.6.8 Identification of the protocol**

This PICS proforma applies to ISA100.11a.

#### **A.1.6.9 Global statement of conformance**

The implementation described in this PICS proforma meets all of the mandatory requirements of the referenced standard:

ISA100.11a:2009

[ ] Yes

[ ] No

NOTE Answering no indicates non-conformance to the specified protocol standard. Non-supported mandatory capabilities are to be identified in the following tables, with an explanation by the implementer explaining why the implementation is non-conforming.

The supplier will have fully complied with the requirements for a statement of conformance by completing the statement contained in this annex. That means that by completing the above, the statement of conformance is complete.

### **A.2 System**

#### **A.2.1 General**

Clause 5 defines the terms field media types and roles. The PICS uses these terms to define mandatory and optional behavior.

#### **A.2.2 Field media type**

Table 429 describes field media types.

**Table 429 – Field media type**

Item number	Device types	Reference	Feature description	Status	Support		
					N/A	Yes	No
A	Field media		PhL & DL	O			

#### **A.2.3 Role implementation**

Table 430 declares which role(s) the device supports along with the mandatory protocol layer and UAP support for that role.

**Table 430 – Protocol layer support**

Item number	Device role	Status					Reference	Support			
		Protocol layers			Medium			N/A	Yes	No	
		AL	TL	NL	Type A	Backbone					
DR1	I/O	M	M	M	M		5.3.2				
DR2	Router	M	M	M	M		5.3.2				
DR3	Backbone router (BBR)	M	M	M	M DR4: O DR7: O	M	5.3.2 5.2.6.3.4 5.2.6.3.4				
DR4	Gateway	M	M	M	DR2:O.1	DR3:O.1	5.3.2				
DR5	System time source	N/A	N/A	N/A	N/A		5.3.2				
DR6	Provisioning	M	M	M	M		5.3.2				
DR7	System manager	M	M	M	DR2:O.2	DR3:O.2	5.3.2				
DR8	Security manager	N/A	N/A	N/A	N/A	N/A	5.3.2				

NOTE This is as noted in 5.2.6.3.4 for devices implementing both BBR and either gateway or system manager roles.

**A.2.4 Device protocol implementation conformance statement**

Table 431 provides the PICS for all devices.

**Table 431 – Device PICS**

Item number	Item description	Reference	Status	Support		
				N/A	Yes	No
DEVSYS-1	Device management application process (DMAP)	5.2.6.1	M			
DEVSYS-2	Device security management function	5.2.6.1	M			
DEVSYS-3	Unique IEEE EUI-64	5.2.7	M			
DEVSYS-4	Unique 128-bit network address assigned by system manager	5.2.7	M			
DEVSYS-5	Ability to report status of device	5.2.8.1	M			
DEVSYS-6	Ability to provide power status attribute to the system manager	5.2.9	M			
DEVSYS-7	Capable of tracking and maintaining network time value to within one (1) s	5.6	M			
DEVSYS-8	Capable of being provisioned using the standard physical interface if that interface is implemented	5.2.6.3.3	M			
DEVSYS-9	Capable of disabling those options defined in this standard	5.2.2	M			
DEVSYS-10	Compliant with specific regional legislation	5.2.55.6.2	M			
DEVSYS-11	Implementation of at least one role	5.2.6.1	M			
DEVSYS-12	Implementation of NL and TL protocol layers	5.2.6.1	M			
DEVSYS-13	Implementation of field media	5.2.6.1	O			
DEVSYS-14	16-bit address assigned by system manager	5.2.7	DEVSYS-13:M			

Table 432 provides the PICS for devices implementing the I/O role.

**Table 432 – PICS for device implementing I/O role**

Item number	Item description	Reference	Status	Support		
				N/A	Yes	No
IOSYS-1	Source and sink data	5.2.6.3.1	M			
IOSYS 2	UAP object	5.2.6.3.1	M			

Table 433 provides the PICS for a routing device.

**Table 433 – PICS for device implementing router role**

Item number	Item description	Reference	Status	Support		
				N/A	Yes	No
RSYS-1	Routing capability	5.2.6.3.2	M			
RSYS-2	Proxy capability	5.2.6.3.2	M			
RSYS-3	Clock propagation	5.2.6.3.2	M			

Table 434 provides the PICS for a backbone router.

**Table 434 – PICS for backbone router**

Item number	Item description	Reference	Status	Support		
				N/A	Yes	No
BRSYS-1	Routing capability via backbone	5.2.6.3.4	M			
BRSYS-2	Proxy capability via backbone	5.2.6.3.4	M			

Table 435 provides the PICS for a gateway.

**Table 435 – PICS for gateway**

Item number	Item description	Reference	Status	Support		
				N/A	Yes	No
GWSYS-1	Communicates over WISN by native client/server access	5.2.6.3.5	O.1			
GWSYS-2	Communicates over WISN by tunneling	5.2.6.3.5	O.1			
GWSYS-3	Interface to foreign protocol (non-WISN)	5.2.6.3.5	M			
GWSYS-4	UAP that communicates over the WISN based on a foreign protocol (non-WISN)	5.2.6.3.5	M			
GWSYS-5	Implement router role		O.2			
GWSYS-6	Implement backbone router role		O.2			

Table 436 provides the PICS for a system manager.

**Table 436 – PICS for system manager**

Item number	Item description	Reference	Status	Support		
				N/A	Yes	No
SMSYS-1	System management application process (SMAP)	5.2.6.3.6	M			
SMSYS-2	Set time source tree	5.2.6.3.6	M			
SMSYS-3	Assign 16 bit DL subnet addresses	5.2.7	M			
SMSYS-4	Implement router role		O.3			
SMSYS-5	Implement backbone router role		O.3			

Table 437 provides the PICS for a provisioning device.

**Table 437 – PICS for provisioning device**

Item number	Item description	Reference	Status	Support		
				N/A	Yes	No
PSYS-1	Provision a device set to factory defaults	5.2.6.3.3	M			
PSYS-2	Device provisioning services object (DPSO)	5.2.6.3.3	M			

Table 438 provides the PICS for a security manager.

**Table 438 – PICS for security manager**

Item number	Item description	Reference	Status	Support		
				N/A	Yes	No
SECSYS-1	Manage security and trust information in the system by working with PSMO in the system manager	5.2.6.3.7	M			

Table 439 provides the PICS for a device implementing the system time source role.

**Table 439 – PICS for device implementing system time source role**

Item number	Item description	Reference	Status	Support		
				N/A	Yes	No
STSSYS-1	Supplies system time	5.2.6.3.8	M			
STSSYS-2	Sense of time: Monotonic increase at rate closely matching real time, accurate to within one second		M			
STSSYS -3	Implement system manager role	5.3.2	O.4			
STSSYS -4	Implement I/O role	5.3.2	O.4			
STSSYS -5	Implement router role	5.3.2	O.4			
STSSYS -6	Implement backbone router role	5.3.2	O.4			
STSSYS -7	Implement gateway role	5.3.2	O.4			

## A.2.5 System protocol implementation conformance statement

Table 440 provides the required roles for a minimal system conforming to this standard.

**Table 440 – System PICS**

Item number	Devices implementing the following roles	Reference	Status	Support		
				N/A	Yes	No
SYS-1	System manager	5.3.2	M			
SYS-2	Security manager	5.3.2	M			
SYS-3	Provisioning	5.3.2	M			
SYS-4	System time source	5.3.2	M			
SYS-5	I/O	5.3.2	M			
SYS-6	Router	5.3.2	O.5			
SYS-7	Backbone router	5.3.2	O.5			
SYS-8	Gateway	5.3.2	O			

NOTE A router or BBR is necessary for the system manager.

## A.3 System management

### A.3.1 Protocol implementation conformance statements

Table 441 provides the PICS for all devices including the device implementing the I/O or gateway roles.

**Table 441 – Device PICS**

Item number	Feature	Reference	Status	Support		
				N/A	Yes	No
DEVSM-1	Capable of joining a network	6.3.9.2	M			
DEVSM-2	Capable of reporting its capabilities to the system manager during the join process	6.3.9.2	M			
DEVSM-3	Capable of leaving the network either by own initiative or by the request of a management entity through reset/reboot or reset to factory defaults	6.3.9.4	M			
DEVSM-4	Implement a DMAP capable of communicating with an SMAP through ASL client/server, publish/subscribe, or alert services	6.2	M			
DEVSM-5	Capable of having its local management objects configured or monitored by a remote management object through TDSAP-0 which corresponds to port number 0xF0B0	6.2.5	M			
DEVSM-6	Capable of scheduled manipulation of management attributes for synchronized cutover	J.2	M			
DEVSM-7	Capable of requesting, utilizing, maintaining and terminating contracts	6.3.11.2	M			
DEVSM-8	Implements the DMO including all attributes, methods and alerts	6.3.9	M			
DEVSM-9	Implements the ARMO including all attributes, methods and alerts; implement support for reporting: Device diagnostic alerts Device communication alerts Process alerts (I/O only) Security alerts	6.2.7.2	M M M M			
DEVSM-10	Implements the DLMO including all attributes, methods and alerts (only when participating in DL subnet)	6.2.8.2	M			
DEVSM-11	Implements the NLMO including all attributes, methods and alerts	6.2.8.2	M			
DEVSM-12	Implements the TLMO including all attributes, methods and alerts	6.2.8.2	M			
DEVSM-13	Implements the ASLMO including all attributes, methods and alerts	6.2.8.2	M			
DEVSM-14	Implements the DPO including all attributes, methods and alerts	6.2.7.6	M			
DEVSM-15	Implements the UDO including all attributes, methods and alerts	6.2.7.3	M			
DEVSM-16	Implements the HRCO including all attributes, methods and alerts	6.2.7.7	M			
DEVSM-17	Implements the DSMO including all attributes, methods and alerts	6.2.7.5	M			

Table 442 provides the PICS for all devices that implement the router or backbone router role.

**Table 442 – Router and backbone router PICS**

Item number	Feature	reference	Status	Support		
				N/A	Yes	No
RTRSM-1	Capable of acting as a proxy for the system manager for other joining devices	6.3.9.2.2	M			
RTRSM-2	Capable of maintaining and propagating time synchronization as a clock repeater based on a system manager established topology	6.3.10	M			
RTRSM-3	Capable of maintaining and propagating time synchronization as a system time source based on a system manager established topology	6.3.10	O			

Table 443 provides the PICS for a device implementing the system time source role.

**Table 443 – PICS for system time source**

Item number	Feature	Reference	Status	Support		
				N/A	Yes	No
STSSM-1	Capable of maintaining and propagating time synchronization as a system time source based on a system manager established topology	6.3.10	M			

Table 444 provides the PICS for the system manager.

**Table 444 – PICS for system manager**

Item number	Feature	Reference	Status	Support		
				N/A	Yes	No
SMSM-1	Implement a SMAP capable of communicating with a DMAP in any compliant device in the network through ASL client/server, publish/subscribe, or alert services Capable of having its local management objects accessed by a remote management object through TDSAP-1 which corresponds to port number 0xF0B1	6.3.2	M			
SMSM-2	Capable of receiving, processing and responding to join requests from field devices and infrastructure devices through implementation of DMSO	6.3.9.5	M			
SMSM-3	Capable of imposing that a compliant device restart or reset to factory defaults	6.3.9.4	M			
SMSM-4	Capable of acting as a proxy to the security manager by implementing all methods and attributes of PSMO	6.3.4	M			
SMSM-5	Capable of scheduled manipulation of management attributes for synchronized cutover	J.2	M			
SMSM-6	Capable of dynamically allocating and assigning addresses to all compliant network devices that join the network – 16-bit DL subnet alias assignment, 128-bit network address assignment	6.3.5.3	M			
SMSM-7	Capable of providing address translation directory service to all devices of the network through implementation of DSO	6.3.5.4	M			
SMSM-8	Capable of providing support for remote firmware upgrade by implementing an UDO that proxies for the device developers host update application	6.3.6	M			
SMSM-9	Capable of receiving alerts from devices in the network by implementing ARO Device diagnostic alerts Device communication alerts Process alerts (I/O only) Security alerts	6.3.7.2	O M O M			
SMSM-10	Capable of offering the following system time services by implementing the STSO: Configuring one or more system time sources in the network Configuring the distribution topology of clock synchronization dissemination through the network Providing UTC time services and maintains UTC adjustments as needed Monitoring time synchronization accuracy	6.3.10	M			
SMSM-11	Capable of intra-device manipulation of management objects attributes and methods; Capable of runtime configuration of attributes of any network devices DMAP objects	6.3.9.3	M			
SMSM-12	Capable of providing contract related services through implementation of SCO: Capable of establishing contract upon requests received from network devices Capable of modifying, renewing and terminating contracts for all network devices Support for contract and message priorities Capable of runtime configuration of management attributes of the contract source and contract destination devices as well as intermediate field and backbone routers resources	6.3.11.2	M			
SMSM-13	Implements the DPSO including all attributes, methods and alerts	6.3.8	O			

Table 445 provides the PICS for a device implementing the provisioning role.

**Table 445 – PICS for provisioning role**

Item number	Feature	Reference	Status	Support		
				N/A	Yes	No
PRSM-1	Implements the DPSO including all attributes, methods and alerts	6.3.8	M			

## A.4 Security manager

### A.4.1 General

Security management is pertinent to seven baseline profiles as defined in Table 481. The seven profiles affect devices that fulfill the roles listed in the role types column in Table 481.

### A.4.2 Protocol implementation conformance statements

Table 446 provides the PICS for all devices implementing the I/O, router, backbone, gateway, or system manager roles.

**Table 446 – Device PICS**

Item number	Feature	Reference	Status	Support		
				N/A	Yes	No
DEVSEC-1	Capable of exchanging cryptographic data with the security manager via the PSMO	0	M			
DEVSEC-2	Capable of exchanging cryptographic information during the join process	7.4	M			
DEVSEC-3	Provides DL sub-layer policy based security services for incoming and outgoing DPDUs by interaction with the security sub-layer Services selectable based on associated DL key DPDU authentication Rejection of messages that are not sourced by device within the network, not received at the intended time or are maliciously replayed (only when participating in a DL subnet)	7.3.2.3  7.3.2.4.10	M  M M			
DEVSEC-4	Capable of decrypting/authenticating all protected DPDUs and TPDUs	7.3.3	M			
DEVSEC-5	Provides TL sub-layer security services for incoming and outgoing TPDUs by interaction with the security sub-layer Services selectable based on associated TL key TPDU authentication Rejection of messages that are not sourced by device in an authorized session, not received at the intended time or are maliciously replayed	7.3.2.5	M  M M			
DEVSEC-6	Provide security statistics collection, threat detection and reporting	7.10	M			

Table 447 provides the PICS for a provisioning role.

**Table 447 – PICS for provisioning role**

Item number	Feature	reference	Status	Support		
				N/A	Yes	No
PSEC-1	Capable of exchanging cryptographic data with the security manager via the PSMO	0	M			
PSEC-2	Capable of exchanging cryptographic information during the join process	7.4	M			
PSEC-3	Capable of provisioning security aspects of new device by writing the join key and the EUI-64 address of the security manager		M			
PSEC-4	Provides DL sub-layer policy based security services for incoming and outgoing DPDUs by interaction with the security sub-layer Services selectable based on associated DL key DPDU authentication Rejection of messages that are not sourced by device within the network, not received at the intended time or are maliciously replayed (only when participating in DL operation)	7.3.2.3 7.3.2.4.10	O O O			
PSEC-5	Capable of decrypting/authenticating all PDUs		M			
PSEC-6	Provides TL sub-layer security services for incoming and outgoing TPDUs by interaction with the security sub-layer Services selectable based on associated TL key TPDU authentication Rejection of messages that are not sourced by device in an authorized session, not received at the intended time or are maliciously replayed	7.3.2.5	M M M			
PSEC-7	Provide security statistics collection, threat detection and reporting	7.10	M			
NOTE PSEC-4 is optional because a provisioning device might not share a DL key with the new device at provisioning time and therefore cannot provide any security services. However, when the provisioning device (clearly distinguished from the provisioning role) joins a network specified in this standard, it behaves as a normal device where DEVSEC-3 is mandatory.						

Table 448 provides the PICS for a security manager.

**Table 448 – PICS for security manager**

Item number	Feature	Reference	Status	Support		
				N/A	Yes	No
SECSEC-1	Capable of interaction with the DSMO in each network devices through the system manager PSMO proxy	7.7.3	M			
SECSEC-2	Capable of authorizing or rejecting devices that attempt to join the network	7.4	M			
SECSEC-3	Capable of key management for both DL and TL security interactions Key archiving	7.7.4	M O			
SECSEC-4	Administers security policies for all network devices	7.8	M			

## A.5 Physical layer

### A.5.1 General

Since this standard's PhL cites the specifications from IEEE Std 802.15.4, the PICS for the PhL reference IEEE Std 802.15.4-2006.

Table 482 describes the physical layer roles.

**Table 449 – PhL roles**

Item number	Item description	IEEE Std 802.15.4 reference	Status	Support		
				N/A	Yes	No
PLD1	The device is a full function device (router or backbone router)	5, 5.2	O.1			
PLD2	The device is a reduced function device (I/O)	5, 5.2	O.1			
O.1: at least one option shall be selected.						

**A.5.2 Protocol implementation conformance statement****A.5.2.1 Physical layer frequency of operation**

Table 450 describes the PhL frequency of operation.

**Table 450 – PhL frequency of operation**

Item number	Item description	IEEE Std 802.15.4 reference	Status	Support		
				N/A	Yes	No
PLRF1	The device operates at a frequency of 2.4 GHz.	5.4.1, Clause 6, Table 1, 6.6.1, 6.5	M			

**A.5.2.2 Physical layer functions**

Table 451 describes the PhL functions.

**Table 451 – PhL functions**

Item number	Item description	IEEE Std 802.15.4 reference	Status	Support		
				N/A	Yes	No
PLF1	Transmission of packets	5.4.1, Clause 6, 6.2.1.1	M			
PLF2	Reception of packets	5.4.1, Clause 6, 6.2.1.2	M			
PLF3	Activation of radio transceiver	5.4.1, Clause 6, 6.2.2.7, 6.2.2.8	M			
PLF4	Deactivation of radio transceiver	5.4.1, Clause 6, 6.2.2.7, 6.2.2.8	M			
PLF5	Energy detection (ED)	5.4.1, Clause 6, 6.2.2.3, 6.2.2.4, 6.9.7	FD1:M O			
PLF6	Link quality indication (LQI)	5.4.1, Clause 6, 6.2.2.3, 6.2.1.2, 6.9.8	M			
PLF7	Channel selection	5.4.1, Clause 6, 6.2.2.3, 6.2.2.9, 6.2.2.10	M			
PLF8	Clear channel assessment (CCA)	5.4.1, Clause 6, 6.2.2.3, 6.2.2.1, 6.9.9	M			
PLF8.1	Mode 1	6.9.9	O.2			
PLF8.2	Mode 2	6.9.9	O.2			
PLF8.3	Mode 3	6.9.9	O.2			
O.2 At least one of these functions shall be supported						

**A.5.2.3 Physical layer packet**

Table 452 describes the PhL packet.

**Table 452 – PhL packet**

Item number	Item description	IEEE Std 802.15.4 reference	Status	Support		
				N/A	Yes	No
PLP1	PhL protocol data unit (PPDU)	6.3	M			

## A.6 Data link layer

#### A.6.1 General

The data link layer (DL) affects four baseline role profiles, as indicated in Table 483.

**Table 453 – DL roles**

Item number	Role types	Reference	Support		
			N/A	Yes	No
DLR0	I/O device				
DLR2	Router				
DLR3	Backbone router				

#### A.6.2 Protocol implementation conformance statement

Table 454, Table 455, and Table 456 provide the DL PICs for various roles.

**Table 454 – DL PICs for device implementing I/O role**

**Table 455 – DL PICs for device implementing router role**

**Table 456 – DL PICS for device implementing backbone router role**

Item number	Role types	Reference	Status	Support		
				N/A	Yes	No
BBRDL-1	Capable of participating in DL subnet		M			
BBRDL-2	Capable of forwarding DL messages along DL graph		O			
BBRDL-3	Capable of discovering routing capable neighbors BBRDL-3.1 Active scan BBRDL-3.2 Passive scan		O O O			
BBRDL-4	Join network via join requests to DL neighbor		X			
BBRDL-5	Capable of advertisement transmission		O			
BBRDL-6	Capable of acting as proxy in joining process		O			
BBRDL-7	Capable of acting as clock master		O			
BBRDL-8	Capable of acting as clock repeater		O			

## A.7 Network layer

### A.7.1 Role profiles and protocol implementation conformance statement

Table 457 describes the PICS for an I/O device.

**Table 457 – PICS for devices implementing I/O role**

Item number	Feature	Reference	Status	Comments	Support		
					N/A	Yes	No
NRNL-1	Network layer headers						
NRNL-1.1	NPDU construction using basic header format	10.5.2	M				
NRNL-1.2	NPDU construction using contract-enabled header	10.5.3	O				
NRNL-1.3	NPDU construction using full header	10.5.4	O				
NRNL-2	Fragmentation and reassembly	10.2.5					
NRNL-2.1	NPDU fragmentation and header construction.	10.5.5	O				
NRNL-2.2	Reassembly process	10.2.5	O				
NRNL-3	Contract table	10.4	M	Contains devices own contracts only			
NRNL-4	Address translation table	10.4	M				
NRNL-5	Routing table	10.4	M	Contains devices own routes only			
NRNL-6	Default route	10.4	O				
NRNL-7	Backbone interface & header expansion	10.5.4	O				

Table 458 describes the PICS for a routing device.

**Table 458 – PICS for device implementing router role**

Item number	Feature	Reference	Status	Comments	Support		
					N/A	Yes	No
NRNL-1	Network layer headers						
NRNL-1.1	NPDU construction using basic header format	10.5.2	M				
NRNL-1.2	NPDU construction using contract-enabled header	10.5.3	O				
NRNL-1.3	NPDU construction using full header	10.5.4	O				
NRNL-2	Fragmentation and reassembly	10.2.5					
NRNL-2.1	NPDU fragmentation and header construction.	10.5.5	O				
NRNL-2.2	Reassembly process	10.2.5	O				
		10.4		Contains devices own contracts only			
NRNL-3	Contract table		M				
NRNL-4	Address translation table	10.4	M				
		10.4		Contains devices own routes only			
NRNL-5	Routing table		M				
NRNL-6	Default route	10.4	O				
NRNL-7	Backbone interface & header expansion	10.5.4	O				

Table 459 describes the PICS for a backbone router.

**Table 459 – PICS for devices implementing backbone router role**

Item number	Feature	Reference	Status	Comments	Support		
					N/A	Yes	No
NRNL-1	Network layer headers						
NRNL-1.1	NPDU construction using basic header format	10.5.2	M				
NRNL-1.2	NPDU construction using contract-enabled header	10.5.3	O				
NRNL-1.3	NPDU construction using full header	10.5.4	O				
NRNL-2	Fragmentation and reassembly	10.2.5					
NRNL-2.1	NPDU fragmentation and header construction.	10.5.5	M				
NRNL-2.2	Reassembly process	10.2.5	M				
NRNL-3	Contract table	10.4	M				
NRNL-4	Address translation table	10.4	M				
NRNL-5	Routing table	10.4	M				
NRNL-6	Default route	10.4	O				
NRNL-7	Backbone interface & header expansion	10.5.4	M				

NOTE The backbone router needs to support more routes at the network layer than either the routing device or I/O device. As such, its address translation tables also need to be capable of supporting more entries. The backbone device by itself need not support more than the minimum set of contracts.

### A.7.2 Provisioning device

No special transport or network layer functions are required for this role. If the provisioning device is a routing device, see PICS for routing device (Table 458). If the PD is a backbone router, see PICS for backbone router (Table 459).

### A.7.3 System time source

No special transport or network layer functions are required for this role. If the time source device is a routing device, see PICS for routing device (Table 458). If the time source is a backbone router, see PICS for backbone router (Table 459).

## A.8 Transport layer

### A.8.1 Role profiles and protocol implementation conformance statement

Table 460 describes the PICS for an I/O device.

**Table 460 – PICS for device implementing I/O role**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
NRTL-1	Data services	11.6.2	M					
NRTL-1.1	Data.request	11.6.2.2	M					
NRTL-1.2	Data.confirm	11.6.2.3	M					
NRTL-1.3	Data.indication	11.6.2.4	M					
NRTL-2	Security services	11.3	M					
NRTL-3	UDP services	11.4	M					
NRTL-3.1	UDP header	11.4	M					
NRTL-3.2	HC_UDP encoding	11.5	M					

Table 461 describes the PICS for a routing device.

**Table 461 – PICS for routing device**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
NRTL-1	Data services	11.6.2	M					
NRTL-1.1	Data.request	11.6.2.2	M					
NRTL-1.2	Data.confirm	11.6.2.3	M					
NRTL-1.3	Data.indication	11.6.2.4	M					
NRTL-2	Security services	11.3	M					
NRTL-3	UDP services	11.4	M					
NRTL-3.1	UDP header	11.4	M					
NRTL-3.2	HC_UDP encoding	11.5	M					

Table 462 describes the PICS for a backbone router.

**Table 462 – PICS for backbone router**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
NRTL-1	Data services	11.6.2	M					
NRTL-1.1	Data.request	11.6.2.2	M					
NRTL-1.2	Data.confirm	11.6.2.3	M					
NRTL-1.3	Data.indication	11.6.2.4	M					
NRTL-2	Security services	11.3	M					
NRTL-3	UDP services	11.4	M					
NRTL-3.1	UDP header	11.4	M					
NRTL-3.2	HC_UDP encoding	11.5	M					

NOTE The backbone router needs to support more routes at the transport layer than either the routing device or I/O device. As such, its address translation tables also need to be capable of supporting more entries. The backbone device by itself need not support more than the minimum set of contracts.

## A.8.2 Provisioning device

No special transport or network layer functions are required for this role. If the provisioning device is a routing device, see PICS for routing device (Table 461). If the PD is a backbone router, see PICS for backbone router (Table 462).

### A.8.3 System time source

No special transport or network layer functions are required for this role. If the time source device is a routing device, see PICS for routing device (Table 461). If the time source is a backbone router, see PICS for backbone router (Table 462).

## A.9 Application layer

### A.9.1 General

Table 463 describes the AL implementation option.

**Table 463 – AL implementation option**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALimpl-1	Supplies download content	12.15.2.4	O					
ALimpl-2	Receives download content	12.15.2.4	O					
ALimpl-3	Supplies upload content	12.15.2.4	O					
ALimpl-4	Receives upload content	12.15.2.4	O					

NOTE 1 An optional feature may become mandatory depending on the role supported.

Table 464 lists supported industry-independent objects.

**Table 464 – PICS part 2: Optional industry-independent objects**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALIO-1	AlertReceiving object	12.15.2.3	O		Indicate number of instances supported in range column.  NOTE 1 Number of object instances supported shall be $0 \leq n \leq 4$  NOTE 2 AlertReceiving object alert categories shall be unique; an AlertReceiving object may not receive alerts of a category assigned to another AlertReceiving object			
ALIO-2	UploadDownload object	12.15.2.4	O		Indicate number of instances supported in range column.			
ALIO-3	Concentrator object	12.15.2.5	O		Indicate number of instances supported in range column.			
ALIO-4	Dispersion object	12.15.2.6	O		Indicate number of instances supported in range column.			
ALIO-5	Tunnel object	12.15.2.7	O		Indicate number of instances supported in the range column.  NOTE Vendor shall indicate protocol(s) supported			
ALIO-6	Interface object	12.15.2.8	O		Indicate number of instances supported in range column.			

Table 465 lists supported standard services for the I/O device role.

**Table 465 – PICS part 2: Supported standard services for I/O device role**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALRD-1	Read service	12.17.4.2.3						
ALRD-1.1	Read.request	12.17.4.2.3.1	O					
ALRD-1.2	Read.indicate	12.17.4.2.3.1	M					
ALRD-1.3	Read.response	12.17.4.2.3.1	M					
ALRD-1.4	Read.confirm	12.17.4.2.3.1	O					
ALWRT-1	Write service	12.17.4.2.4	O					
ALWRT-1.1	Write.request	12.17.4.2.4.1						
ALWRT-1.2	Write.indicate	12.17.4.2.4.1	M					
ALWRT-1.3	Write.response	12.17.4.2.4.1	M					
ALWRT-1.4	Write.confirm	12.17.4.2.4.1	O					
ALEX-1	Execute service	12.17.4.2.5						
ALEX-1.1	Execute.request	12.17.4.2.5.1	O					
ALEX-1.2	Execute.indicate	12.17.4.2.5.1	M					
ALEX-1.3	Execute.response	12.17.4.2.5.1	M					
ALEX-1.4	Execute.confirm	12.17.4.2.5.1	O					
ALPUB-1	Publish service	12.17.3.2						
ALPUB-1.1	Publish.request	12.17.3.2.1	M					
ALPUB-1.2	Publish.indicate	12.17.3.2.1	O					
ALALRTRPT-1	AlertReport service	12.17.5.2.2						
ALALRTRPT-1.1	AlertReport.request	12.17.5.2.2.1	M					
ALALRTRPT-1.2	AlertReport.indicate	12.17.5.2.2.1	O					
ALALRTACK	AlertAcknowledge service	12.17.5.2.3						
ALALRTACK-1.1	AlertAcknowledge.request	12.17.5.2.3.1	O					
ALALRTACK-1.2	AlertAcknowledge.indicate	12.17.5.2.3.1	M					
ALTUNL-1	Tunnel service	12.17.6.2						
ALTUNL-1.1	Tunnel.request	12.17.6.2.1	O					
ALTUNL-1.2	Tunnel.response	12.17.6.2.1	O					
ALTUNL-1.3	Tunnel.indicate	12.17.6.2.1	O					
ALTUNL-1.4	Tunnel.confirm	12.17.6.2.1	O					

Table 466 lists supported standard services for the system manager role.

**Table 466 – PICS part 2: Supported standard services for system manager role**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALRD-1	Read service	12.17.4.2.3						
ALRD-1.1	Read.request	12.17.4.2.3.1	M					
ALRD-1.2	Read.indicate	12.17.4.2.3.1	O					
ALRD-1.3	Read.response	12.17.4.2.3.1	O					
ALRD-1.4	Read.confirm	12.17.4.2.3.1	M					
ALWRT-1	Write service	12.17.4.2.4						
ALWRT-1.1	Write.request	12.17.4.2.4.1	M					
ALWRT-1.2	Write.indicate	12.17.4.2.4.1	O					
ALWRT-1.3	Write.response	12.17.4.2.4.1	O					
ALWRT-1.4	Write.confirm	12.17.4.2.4.1	M					
ALEX-1	Execute service	12.17.4.2.5						
ALEX-1.1	Execute.request	12.17.4.2.5.1	M					
ALEX-1.2	Execute.indicate	12.17.4.2.5.1	O					
ALEX-1.3	Execute.response	12.17.4.2.5.1	O					
ALEX-1.4	Execute.confirm	12.17.4.2.5.1	M					
ALPUB-1	Publish service	12.17.3.2						
ALPUB-1.1	Publish.request	12.17.3.2.1	O					
ALPUB-1.2	Publish.indicate	12.17.3.2.1	O					
ALALRTRPT-1	AlertReport service	12.17.5.2.2						
ALALRTRPT-1.1	AlertReport.request	12.17.5.2.2.1	O					
ALALRTRPT-1.2	AlertReport.indicate	12.17.5.2.2.1	M					
ALALRTACK	AlertAcknowledge service	12.17.5.2.3						
ALALRTACK-1.1	AlertAcknowledge.request	12.17.5.2.3.1	M					
ALALRTACK-1.2	AlertAcknowledge.indicate	12.17.5.2.3.1	O					
ALTUNL-1	Tunnel service	12.17.6.2						
ALTUNL-1.1	Tunnel.request	12.17.6.2.1	O					
ALTUNL-1.2	Tunnel.response	12.17.6.2.1	O					
ALTUNL-1.3	Tunnel.indicate	12.17.6.2.1	O					
ALTUNL-1.4	Tunnel.confirm	12.17.6.2.1	O					

Table 467 lists supported standard services for a gateway role.

**Table 467 – PICS part 2: Supported standard services for gateway role when supporting native access**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
SMREMON11a	System manager is remote from the gateway, and is communicating with the gateway over the WISN		O					
ALRD-1	Read service	12.17.4.2.3						
ALRD-1.1	Read.request	12.17.4.2.3.1	M					
ALRD-1.2	Read.indicate	12.17.4.2.3.1	O SMREMON11a : M					
ALRD-1.3	Read.response	12.17.4.2.3.1	O SMREMON11a : M					
ALRD-1.4	Read.confirm	12.17.4.2.3.1	M					
ALWRT-1	Write service	12.17.4.2.4						
ALWRT-1.1	Write.request	12.17.4.2.4.1	M					
ALWRT-1.2	Write.indicate	12.17.4.2.4.1	O SMREMON11a : M					
ALWRT-1.3	Write.response	12.17.4.2.4.1	O SMREMON11a : M					
ALWRT-1.4	Write.confirm	12.17.4.2.4.1	M					
ALEX-1	Execute service	12.17.4.2.5						
ALEX-1.1	Execute.request	12.17.4.2.5.1	M					
ALEX-1.2	Execute.indicate	12.17.4.2.5.1	O SMREMON11a : M					
ALEX-1.3	Execute.response	12.17.4.2.5.1	O SMREMON11a : M					
ALEX-1.4	Execute.confirm	12.17.4.2.5.1	M					
ALPUB-1	Publish service	12.17.3.2						
ALPUB-1.1	Publish.request	12.17.3.2.1	O SMREMON11a : M					
ALPUB-1.2	Publish.indicate	12.17.3.2.1	O					
ALALRTRPT-1	AlertReport service	12.17.5.2.2						
ALALRTRPT-1.1	AlertReport.request	12.17.5.2.2.1	O SMREMON11a : M					
ALALRTRPT-1.2	AlertReport.indicate	12.17.5.2.2.1	O					
ALALRTACK	AlertAcknowledge service	12.17.5.2.3						
ALALRTACK-1.1	AlertAcknowledge.request	12.17.5.2.3.1	O					
ALALRTACK-1.2	AlertAcknowledge.indicate	12.17.5.2.3.1	O SMREMON11a : M					
ALTUNL-1	Tunnel service	12.17.6.2						
ALTUNL-1.1	Tunnel.request	12.17.6.2.1	O					
ALTUNL-1.2	Tunnel.response	12.17.6.2.1	O					
ALTUNL-1.3	Tunnel.indicate	12.17.6.2.1	O					
ALTUNL-1.4	Tunnel.confirm	12.17.6.2.1	O					

Table 468 lists supported standard services for a tunneling gateway role.

**Table 468 – PICS part 2: Supported standard services for gateway role when supporting interoperable tunneling and for adapters**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
SMREMON11a	System manager is remote from the gateway, and is communicating with the gateway over the WISN	Clause 6	O					
ALRD-1	Read service	12.17.4.2.3						
ALRD-1.1	Read.request	12.17.4.2.3.1	O					
ALRD-1.2	Read.indicate	12.17.4.2.3.1	O SMREMON11a : M					
ALRD-1.3	Read.response	12.17.4.2.3.1	O SMREMON11a : M					
ALRD-1.4	Read.confirm	12.17.4.2.3.1	O					
ALWRT-1	Write service	12.17.4.2.4						
ALWRT-1.1	Write.request	12.17.4.2.4.1	O					
ALWRT-1.2	Write.indicate	12.17.4.2.4.1	O SMREMON11a : M					
ALWRT-1.3	Write.response	12.17.4.2.4.1	O SMREMON11a : M					
ALWRT-1.4	Write.confirm	12.17.4.2.4.1	O					
ALEX-1	Execute service	12.17.4.2.5						
ALEX-1.1	Execute.request	12.17.4.2.5.1	O					
ALEX-1.2	Execute.indicate	12.17.4.2.5.1	O SMREMON11a : M					
ALEX-1.3	Execute.response	12.17.4.2.5.1	O SMREMON11a : M					
ALEX-1.4	Execute.confirm	12.17.4.2.5.1	O					
ALPUB-1	Publish service	12.17.3.2						
ALPUB-1.1	Publish.request	12.17.3.2.1	O SMREMON11a : M					
ALPUB-1.2	Publish.indicate	12.17.3.2.1	O					
ALALRTRPT-1	AlertReport service	12.17.5.2.2						
ALALRTRPT - 1.1	AlertReport.request	12.17.5.2.2.1	O SMREMON11a : M					
ALALRTRPT - 1.2	AlertReport.indicate	12.17.5.2.2.1	O					
ALALRTACK	AlertAcknowledge service	12.17.5.2.3						
ALALRTACK-1.1	AlertAcknowledge.request	12.17.5.2.3.1	O					
ALALRTACK-1.2	AlertAcknowledge.indicate	12.17.5.2.3.1	O SMREMON11a : M					
ALTUNL-1	Tunnel service	12.17.6.2						
ALTUNL-1.1	Tunnel.request	12.17.6.2.1	M					
ALTUNL-1.2	Tunnel.response	12.17.6.2.1	O					
ALTUNL-1.3	Tunnel.indicate	12.17.6.2.1	M					
ALTUNL-1.4	Tunnel.confirm	12.17.6.2.1	O					

Table 469 lists supported standard services for a routing device role.

**Table 469 – PICS part 2: Supported standard services for routing device role**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALRD-1	Read service	12.17.4.2.3						
ALRD-1.1	Read.request	12.17.4.2.3.1	O					
ALRD-1.2	Read.indicate	12.17.4.2.3.1	M					
ALRD-1.3	Read.response	12.17.4.2.3.1	M					
ALRD-1.4	Read.confirm	12.17.4.2.3.1	O					
ALWRT-1	Write service	12.17.4.2.4						
ALWRT-1.1	Write.request	12.17.4.2.4.1	O					
ALWRT-1.2	Write.indicate	12.17.4.2.4.1	M					
ALWRT-1.3	Write.response	12.17.4.2.4.1	M					
ALWRT-1.4	Write.confirm	12.17.4.2.4.1	O					
ALEX-1	Execute service	12.17.4.2.5						
ALEX-1.1	Execute.request	12.17.4.2.5.1	O					
ALEX-1.2	Execute.indicate	12.17.4.2.5.1	M					
ALEX-1.3	Execute.response	12.17.4.2.5.1	M					
ALEX-1.4	Execute.confirm	12.17.4.2.5.1	O					
ALPUB-1	Publish service	12.17.3.2						
ALPUB-1.1	Publish.request	12.17.3.2.1	M					
ALPUB-1.2	Publish.indicate	12.17.3.2.1	O					
ALALRTRPT-1	AlertReport service	12.17.5.2.2						
ALALRTRPT-1.1	AlertReport.request	12.17.5.2.2.1	M					
ALALRTRPT-1.2	AlertReport.indicate	12.17.5.2.2.1	O					
ALALRTACK	AlertAcknowledge service	12.17.5.2.3						
ALALRTACK-1.1	AlertAcknowledge.request	12.17.5.2.3.1	O					
ALALRTACK-1.2	AlertAcknowledge.indicate	12.17.5.2.3.1	M					

Table 470 lists supported standard services for a backbone router role.

**Table 470 – PICS part 2: Supported standard services for backbone router role**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALRD-1	Read service	12.17.4.2.3	O					
ALRD-1.1	Read.request	12.17.4.2.3.1	O					
ALRD-1.2	Read.indicate	12.17.4.2.3.1						
ALRD-1.3	Read.response	12.17.4.2.3.1	O					
ALRD-1.4	Read.confirm	12.17.4.2.3.1	O CFGVIAWIS N: M					
ALWRT-1	Write service	12.17.4.2.4	O CFGVIAWIS N: M					
ALWRT-1.1	Write.request	12.17.4.2.4.1	O					
ALWRT-1.2	Write.indicate	12.17.4.2.4.1						
ALWRT-1.3	Write.response	12.17.4.2.4.1	O					
ALWRT-1.4	Write.confirm	12.17.4.2.4.1	O CFGVIAWIS N: M					
ALEX-1	Execute service	12.17.4.2.5	O CFGVIAWIS N: M					
ALEX-1.1	Execute.request	12.17.4.2.5.1	O					
ALEX-1.2	Execute.indicate	12.17.4.2.5.1						
ALEX-1.3	Execute.response	12.17.4.2.5.1	O					
ALEX-1.4	Execute.confirm	12.17.4.2.5.1	O CFGVIAWIS N: M					
ALPUB-1	Publish service	12.17.3.2	O CFGVIAWIS N: M					
ALPUB-1.1	Publish.request	12.17.3.2.1	O CFGVIAWIS N: M					
ALPUB-1.2	Publish.indicate	12.17.3.2.1						
ALALRTRPT-1	AlertReport service	12.17.5.2.2	O					
ALALRTRPT-1.1	AlertReport.request	12.17.5.2.2.1	O					
ALALRTRPT-1.2	AlertReport.indicate	12.17.5.2.2.1						
ALALRTACK	AlertAcknowledge service	12.17.5.2.3	O ALRTS2WIS NSM : M					
ALALRTACK-1.1	AlertAcknowledge.request	12.17.5.2.3.1	O					
ALALRTACK-1.2	AlertAcknowledge.indicate	12.17.5.2.3.1						

Table 471 lists supported standard services for the provisioning role.

**Table 471 – PICS part 2: Supported standard services for provisioning role**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALRD-1	Read service	12.17.4.2.3						
ALRD-1.1	Read.request	12.17.4.2.3.1	M					
ALRD-1.2	Read.indicate	12.17.4.2.3.1	O					
ALRD-1.3	Read.response	12.17.4.2.3.1	O					
ALRD-1.4	Read.confirm	12.17.4.2.3.1	M					
ALWRT-1	Write service	12.17.4.2.4						
ALWRT-1.1	Write.request	12.17.4.2.4.1	M					
ALWRT-1.2	Write.indicate	12.17.4.2.4.1	O					
ALWRT-1.3	Write.response	12.17.4.2.4.1	O					
ALWRT-1.4	Write.confirm	12.17.4.2.4.1	M					
ALEX-1	Execute service	12.17.4.2.5						
ALEX-1.1	Execute.request	12.17.4.2.5.1	M					
ALEX-1.2	Execute.indicate	12.17.4.2.5.1	O					
ALEX-1.3	Execute.response	12.17.4.2.5.1	O					
ALEX-1.4	Execute.confirm	12.17.4.2.5.1	M					

Table 472 lists supported standard services for the system time source role.

**Table 472 – PICS part 2: Supported standard services for system time source role**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALRD-1	Read service	12.17.4.2.3						
ALRD-1.1	Read.request	12.17.4.2.3.1	O					
ALRD-1.2	Read.indicate	12.17.4.2.3.1	M					
ALRD-1.3	Read.response	12.17.4.2.3.1	M					
ALRD-1.4	Read.confirm	12.17.4.2.3.1	O					
ALWRT-1	Write service	12.17.4.2.4						
ALWRT-1.1	Write.request	12.17.4.2.4.1	O					
ALWRT-1.2	Write.indicate	12.17.4.2.4.1	M					
ALWRT-1.3	Write.response	12.17.4.2.4.1	M					
ALWRT-1.4	Write.confirm	12.17.4.2.4.1	O					
ALEX-1	Execute service	12.17.4.2.5						
ALEX-1.1	Execute.request	12.17.4.2.5.1	O					
ALEX-1.2	Execute.indicate	12.17.4.2.5.1	M					
ALEX-1.3	Execute.response	12.17.4.2.5.1	M					
ALEX-1.4	Execute.confirm	12.17.4.2.5.1	O					

#### A.9.2 Process control profile supported objects

Table 473 lists supported objects for process control conformance.

**Table 473 – Process control conformance: Supported objects**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALPCO-1	Analog input	12.19.7.3	O		Use Range column for number of instances supported			
ALPCO-2	Analog output	12.19.7.4	O		Use Range column for number of instances supported			
ALPCO-3	Digital input	12.19.7.5	O		Use Range column for number of instances supported			
ALPCO-4	Digital output	12.19.7.6	O		Use Range column for number of instances supported			

### A.9.3 Process control profile supported alerts

Table 474 lists supported alerts for process control conformance.

**Table 474 – Process control conformance: Supported alerts**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
ALPCAIALRT-1	Analog input	12.19.7.3	O					
ALPCAIALRT-1.1	Analog input High alarm	12.22.2.10 and 12.23.2	O					
ALPCAIALRT-1.2	Analog input High High alarm	12.22.2.10 and 12.23.2	O					
ALPCAIALRT-1.3	Analog input Low alarm	12.22.2.10 and 12.23.2	O					
ALPCAIALRT-1.4	Analog input Low Low alarm	12.22.2.10 and 12.23.2	O					
ALPCAIALRT-1.5	Analog input Deviation High alarm	12.22.2.10 and 12.23.2	O					
ALPCAIALRT-1.6	Analog input Deviation Low alarm	12.22.2.10 and 12.23.2	O					
ALPCAIALRT-1.7	Analog input Out of Service alarm	12.22.2.10 and 12.23.2	O					
ALPCAOALRT-1	Analog output	12.19.7.4	O					
ALPCAOALRT-1.1	Analog output High alarm	12.22.2.10 and 12.23.2	O					
ALPCAOALRT-1.2	Analog output High High alarm	12.22.2.10 and 12.23.2	O					
ALPCAOALRT-1.3	Analog output Low alarm	12.22.2.10 and 12.23.2	O					
ALPCAOALRT-1.4	Analog output Low Low alarm	12.22.2.10 and 12.23.2	O					
ALPCAOALRT-1.5	Analog output Deviation High alarm	12.22.2.10 and 12.23.2	O					
ALPCAOALRT-1.6	Analog output Deviation Low alarm	12.22.2.10 and 12.23.2	O					
ALPCAOALRT-1.7	Analog output Out of Service alarm	12.22.2.10 and 12.23.2	O					
ALPCBIALRT-1	Binary input	12.19.7.5	O					
ALPCBIALRT-1.1	Binary input Discrete alarm	12.22.2.10 and 12.23.2	O					
ALPCBIALRT-1.2	Binary input Out of Service alarm	12.22.2.10 and 12.23.2	O					
ALPCBOALRT-1	Binary output	12.19.7.6	O					
ALPCBOALRT-1.1	Binary output Discrete alarm	12.22.2.10 and 12.23.2	O					
ALPCBOALRT-1.2	Binary output Out of Service alarm	12.22.2.10 and 12.23.2	O					

### A.10 Gateway

Table 475 provides PICS for a gateway.

**Table 475 – PICS: Gateway**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
GWGA-1	UAP	Clause 13	M	At least 1				
GWGA-1.1	4-bit object addressing	Clause 13	M					
GWGA-1.2	8-bit object addressing	Clause 13	O					
GWGA-1.3	16-bit object addressing	Clause 13	O					
GWGA-2	Native client/server access	13.3.4	O					
GWGA-2.1	IFO	13.3.4	M	At least 1				
GWGA-2.2	Buffered message behavior	13.3.4	M					
GWGA-2.3	Supports upload and download	13.3.1.10	O					
GWGA-3	Native publish/subscribe access	13.3.4	O					
GWGA-3.1	CON	13.3.4	O	At least 1				
GWGA-3.2	CON buffered message behavior	13.3.4	M					
GWGA-3.3	DIS	13.3.4	O	At least 1				
GWGA-3.4	DIS buffered message behavior	13.3.4	M					
GWGA-4	Tunneling	13.3.1	O					
GWGA-4.1	TUN	13.3.1.2	M	At least 2				
GWGA-4.2	Supports a foreign protocol	Annex O	M	At least 1				
GWGA-4.3	Supports 2-part tunnel	13.3.1.4	M					
GWGA-4.4	Supports 4-part tunnel	13.3.1.4	O					
GWGA-4.5	Supports publish	13.3.1.4	O					
GWGA-4.6	Supports subscribe	13.3.1.4	O					
GWGA-4.7	Supports peer TUNs	13.3.1.5	M	At least 1				
GWGA-4.8	TUN non-buffered message behavior	13.3.1.4	M					
GWGA-4.9	TUN buffered message behavior	13.3.1.4 13.3.1.6.2	O					
GWGA-4.10	Foreign_Destination_Address	13.3.1.8	O	At least 1 octet				
GWGA-4.11	Foreign_Source_Address	13.3.1.8	O	At least 1 octet				
GWGA-4.12	Connection_Info[ ]	13.3.1.9	O	At least 1 octet				
GWGA-4.13	Transaction_Info[ ]	13.3.1.9	O	At least 1 octet				
GWGA-4.14	7-bit payload length	13.3.1.6.1	M					
GWGA-4.15	15-bit payload length	13.3.1.6.1	O					
GWGA-5	System manager access	13.1.3	M					
GWGA-5.1	Embedded system manager	13.1.3	O					
GWGA-5.2	Configurable external reference to system manager	13.1.3	M					
GWGA-6	Alerts	13.3.3	O					
GWGA-6.1	ARO	13.1.3	M					
GWGA-6.2	Accepts process alerts	13.1.3	O					
GWGA-6.3	Accepts device alerts	13.1.3	O					
GWGA-6.4	Accepts network alerts	13.1.3	O					
GWGA-6.5	Accepts security alerts	13.1.3	O					
GWGA-6.6	Acknowledges alerts	13.1.3	M					
GWGA-6.7	Supports alert cascading	13.1.3	O					
GWGA-6.8	ARMO	13.1.3	O					
GWGA-6.9	Generates process alerts	13.1.3	O					
GWGA-6.10	Generates device alerts	13.1.3	O					
GWGA-6.11	Generates network alerts	13.1.3	O					
GWGA-6.12	Generates security alerts	13.1.3	O					
GWGA-6.13	Supports alarm recovery	13.1.3	M					
GWGA-7	Time management	13.3.5	O					
GWGA-7.1	Propagates time through an embedded radio interface	13.3.5	O.1					
GWGA-7.2	Propagates time through a BBR	13.3.5	O.1					
GWGA-8	Security manager access	13.1.3	M					

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
GWGA-8.1	Embedded security manager	13.1.3	O					
GWGA-8.2	Configurable external reference to security manager	13.1.3	M					
GWGA-9	Radio interface	13.1.3	O					
GWGA-9.1	This standard	13.1.3	O					
GWGA-9.2	Other standard	13.1.3	O					
GWGA-10	Backbone interface	13.1.3	O					
GWGA-11	UAPMO attributes	Table 426	M					
GWGA-11.1	Min Devices	Table 426	M	At least 5				
GWGA-11.2	Min Leases	Table 426	M	2 x NSD - 3	NSD >= 5			
GWGA-12	Ability to join network	7.4	M					
GWGA-12.1	With system manager in gateway	7.4	O.5					
GWGA-12.2	With system manager across backbone	7.4	O.5					
GWGA-12.3	With system manager across Type A field media	7.4	O.5					
GWGA-13	Multimode gateway coexistence	13.2	O					
GWGA-13.1	Supports Session	13.2	M					
GWGA-13.2	Supports Device_List_Report	13.2	M					
GWGA-13.3	Supports Topology_Report	13.2	M					
GWGA-13.4	Supports Schedule_Report	13.2	M					
GWGA-13.5	Supports Device_Health_Report	13.2	M					
GWGA-13.6	Supports Neighbor_Health_Report	13.2	M					
GWGA-13.7	Supports Time	13.2	M					
GWGA-14	Multimode gateway interoperability support	13.2	O					
GWGA-14.1	Supports Session	13.2	M					
GWGA-14.2	Supports Device_List_Report	13.2	M					
GWGA-14.3	Supports Time	13.2	O					
GWGA-14.4	Supports Client/Server	13.2	O.6					
GWGA-14.5	Supports Publish/Subscribe	13.2	O.6					
GWGA-14.6	Supports Bulk_Transfer	13.2	O.6					
GWGA-14.7	Supports Alert	13.2	O.6					

## A.11 Provisioning

Table 476 provides the PICS for I/O devices, routing devices, gateways, and backbone routers, all devices with the Type A field medium.

**Table 476 – PICS: I/O devices, routing devices, gateways, and backbone routers**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
DBPR-1	Joining a provisioning network using Key_Global = ISA100	14.6	M					
DBPR-2	Joining a provisioning network using Key_Join = OPEN	14.6	O		Default value of Key_Join = OPEN. Disabled once S is overwritten. Enabled again only if device reset to factory defaults			
DBPR-3	Support for (out of band) OOB provisioning	14.6	O		Enabled by default if OOB means available			
DBPR-4	Support for asymmetric cryptography-based provisioning	14.6	O					
DBPR-5	Ability to reset device to factory defaults	14.6	M					
DBPR-6	Support for multiple PKI certificates	14.10	O					

Table 477 provides the PICS for provisioning devices.

**Table 477 – PICS: Provisioning devices**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
PDPR-1	Support for join/provisioning using Key_Global = ISA100	14.6	M		Only network information and device configuration provisioned. Trust-related information provisioned only with asymmetric crypto			
PDPR-2	Support for join/provisioning using Key_Join = OPEN	14.6	O		Disabled by default. When enabled, both trust-related and network information can be provisioned			
PDPR-3	Support for OOB provisioning	14.5	O		Enabled by default if OOB means available			
PDPR-4	Support for asymmetric cryptography-based provisioning	14.6	O					
PDPR-5	Support for device-specific provisioning	14.10	O					

## **Annex B (NORMATIVE)**

### **Role profiles**

#### **B.1 Introduction**

##### **B.1.1 General**

A role profile is defined as the baseline capabilities, including any options, settings, and configurations, that are required of a device to perform that role adequately. The roles are defined in 5.2.6.2, but are listed for reference here as system manager, security manager, backbone router, router, I/O, gateway, system time source, and provisioning device.

##### **B.1.2 Scope**

This annex provides the role profile proforma for compliance to this standard.

##### **B.1.3 Purpose**

The role profile will define those device capabilities such as options, settings, and configurations necessary to fulfill each specific role defined in 5.2.6.2. The purpose for this is to ensure that devices complying with this standard including this annex will be interoperable. This information is complementary to the PICS.

##### **B.1.4 System size**

The capabilities required of a device to implement a role may be dependent upon the number of devices in the intended system. The minimum system size is defined in Clause 5, but there is no maximum system size. To allow the requirements of this annex to serve a broad range of system sizes, those requirements dependent upon system size shall use a formula to specify the minimum capability. For the purposes of this annex, the number of system devices is referred to as NSD.

##### **B.1.5 Abbreviations and special symbols**

Abbreviations and symbols used include:

- Notations for requirement status:
- M-Mandatory
- O- Optional
- O.n - Optional, but support of at least one of the group of options labeled O.n is required.
- N/A- Not applicable
- X - Prohibited
- Item: Conditional, status dependent upon the support marked for the item

For example, a status of FD1:O.1 and FD2:O.1 indicates that the status is optional but at least one of the features described in FD1 and FD2 is required.

##### **B.1.6 Role profiles**

Table 478 describes the protocol layers and media requirements for all role profiles. Should a device be declared to support more than one role, that device shall fulfill minimum capabilities for each role declared.

**Table 478 – Protocol layer device roles**

Item number	Device role	Status					Reference	Support			
		Protocol layers			Medium			N/A	Yes	No	
		AL	TL	NL	Type A	Backbone					
DR1	I/O	M	M	M	M	N/A	5.2.6.3.1				
DR2	Router	M	M	M	M	N/A	5.2.6.3.2				
DR3	Backbone router	M	M	M	M DR4: O DR7: O	M	5.2.6.3.4 5.2.6.3.4 5.2.6.3.4				
DR4	Gateway	M	M	M	DR2:O.1	DR3:O.1	5.2.6.3.5				
DR5	System time source	N/A	N/A	N/A	N/A	N/A	5.2.6.3.8				
DR6	Provisioning	M	M	M	M	N/A	5.2.6.3.3				
DR7	System manager	M	M	M	DR2:O.2	DR3:O.2	5.2.6.3.6				
DR8	Security manager	N/A	N/A	N/A	N/A	N/A	5.2.6.3.7				

## B.2 System

The protocol of WISN supports the ability to upgrade devices over the air, as shown in Table 479.

**Table 479 – Over-the-air upgrades**

Item number	Role types affected	Reference	Status	Support		
				N/A	Yes	No
OTAR1	I/O		M			
OTAR2	Router		M			
OTAR3	Backbone router		N/A			
OTAR4	Gateway		N/A			
OTAR5	System manager		N/A			
OTAR6	Provisioning device		O			

## B.3 System manager

### B.3.1 General

The system manager allocates the ability for devices to communicate by generating, distributing, and maintaining contracts that define the resources necessary for that communication need. Since each device is required to store its contracts, the capacity of a device for contract storage is critical. While the necessary capacities of the I/O, router, and backbone router devices are dependent upon the number of application objects within those devices, the gateway and system manager are dependent upon the number of devices in the system, defined in this annex as NSD. NSD does not include the system manager in its device count. To allow the requirement of supported contracts to scale with the number of devices in the system, the minimum number of contracts supported for the gateway and system manager is a formula.

Contracts require communication sessions for communication, established by the security manager in conjunction with the system manager. Multiple contracts, communicating to the same endpoints, may share a single session. Minimum capacities described here assume that each session is matched with a single contract, recognizing that more contracts may be needed depending on the nature of the device's applications.

## B.4 Security manager

The security manager establishes sessions between application processes. For example, when a device joins the network it needs a DMAP-SMAP session. The number of sessions that a device implementing a role shall be able to maintain is defined in Table 480. The

number of sessions supported by a system manager is dependent on NSD. The number of keys supported by a gateway is dependent on the number of Gateway-UAP connections that the gateway is designed to support, referred to as GUC in Table 480.

An I/O device is presumed to require capacity to support the following sessions:

- A session between the device's DMAP and the SMAP, established when the device joins the network.
- A session between the device's UAP and a first device such as a gateway.
- A session between the device's DMAP and the first device, for reporting process alerts.
- A session between the device's UAP and a second device's UAP, such as for peer-to-peer communication.

**Table 480 – Session support profiles**

Item number	Role types affected	Minimum # sessions supported	Comments	Status	Support		
					N/A	Yes	No
NCS1	I/O	4	DMAP-SMAP UAP-Gateway DMAP-Gateway UAP-Peer	M			
NCS2	Router	1	DMAP-SMAP	M			
NCS3	Backbone router	1	DMAP-SMAP	M			
NCS4	Gateway	$=(2 \times \text{GUC}) + 1$	DMAP-SMAP GUC x (Gateway-UAP) GUC x (Gateway-DMAP)	M			
NCS5	System manager	NSD	NSD x (SMAP-DMAP)	M			

The security manager assigns the security keys that are required for communication between devices. The number of keys that a device implementing a role shall be able to maintain is defined in Table 481. The number of keys supported for a device depends on the number of sessions supported, with minimum capacities shown in Table 480. In addition, each device needs capacity for a join key, a master key, and a DL key if a DL is included on the device. Key counts need to be doubled, because all keys except for the join key may be in the process of change-over.

**Table 481 – Baseline profiles**

Item number	Role types affected	Minimum # keys supported	Comments	Reference	Status	Support		
						N/A	Yes	No
NKS1	I/O	$1+((\text{NCS1}+2)*2)$		7.2.2	M			
NKS2	Router	$1+((\text{NCS2}+2)*2)$		7.2.2	M			
NKS3	Backbone router	$1+((\text{NCS3}+2)*2)$		7.2.2	M			
NKS4	Gateway	$1+((\text{NCS4}+1)*2)$	Add 2 if gateway has a DL.	7.2.2	M			
NKS5	System manager	$(\text{NCS5}+1)*2$	Add 2 if SM has a DL					
NKS7	Security manager	-N/A		7.2.2	N/A			

## B.5 Physical layer

Since the PhL cites the specifications from IEEE Std 802.15.4, the role capabilities for the PhL reference IEEE Std 802.15.4.

Table 482 describes the physical layer roles.

**Table 482 – PhL roles**

Item number	Item description		IEEE Std 802.15.4 reference	Status	Support		
					N/A	Yes	No
PLR1	I/O	The device is a reduced function device	5, 5.2	O.1			
		The device is a full function device	5, 5.2	O.1			
PLR2	Router	The device is a full function device	5, 5.2	M			
PLR3	Backbone router	The device is a full function device	5, 5.2	M			
PLR4	Provisioning device	The device is a full function device	5, 5.2	M			
O.1: at least one option shall be selected.							

## B.6 Data link layer

### B.6.1 General

The data link layer (DL) affects four role profiles, as indicated in Table 483.

**Table 483 – DL required for listed roles**

Item number	Role types	Reference	Status	Support		
				N/A	Yes	No
DLR1	I/O	5.2.6.3.1	M			
DLR2	Router	5.2.6.3.2	M			
DLR3	Backbone router	5.2.6.3.4	M			
DLR4	Provisioning	5.2.6.3.3	M			

### B.6.2 Role profiles

#### B.6.2.1 General

A DL role profile describes a set of minimum capabilities that shall be supported by every compliant device that implements the Type A field medium. For example, a device filling the router role shall support 8 neighbors. If a device meets all of the other requirements of a router, but supports only 4 neighbors, it is not compliant in its role as router. A device may exceed any of the requirements of its role, as long as all of the roles minimum requirements are met.

The DL is configured through settings to DL management object (DLMO) attributes, and the various roles are described as ranges of DLMO settings that a device can support.

#### B.6.2.2 Data link layer management object attributes

A device's level of support for a capability can be expressed in relation to a set of DLMO attributes and elements of those attributes. This subclause enumerates each attribute and/or element with support that varies by role.

**NOTE** If a number or range of numbers is listed, then a device filling this role shall support that number. If a single number is listed, it shall be interpreted as a minimum value unless indicated otherwise. For example, if a device shall support 3 neighbors, then it may support 4 neighbors, but is non-compliant if it supports only 2 neighbors. An I/O device may be capable of routing even if it isn't fully compliant with the router role; hence some capabilities related to routing are shown as optional (not prohibited) for an I/O device.

Table 484 describes simple DLMO attributes with a single element. (Subsequent tables address DLMO attributes containing multiple elements.)

**Table 484 – Role profiles: General DLMO attributes**

Attribute	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
ActScanHostFract	O	M	M	A non-mains device will not necessarily have the energy to act as an active scanning host for an extended period of time. See Table 485			
AdvJoinInfo AdvSuperframe	O	M	M	All routers and backbone routers can be configured to send advertisements			
TaiTime TaiAdjust	M	M	M	The DL is not necessarily the source of TaiTime for a particular device, and there are cases where a device's DL might not be involved in time propagation as a source or recipient. For example, a BBR might remain time synchronized through a backbone mechanism, and not be involved in DL time propagation			
ClockTimeout	M	M	M	A BBR may be configured as a clock recipient, but this is not intended as typical.			

Table 485 describes baseline role profiles for the dlmo.Device\_Capability attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 485 – Role profiles: dimo.Device\_Capability**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
QueueCapacity	-0	10	20	System manager configures the DL queue only to the extent that the device is forwarding messages on behalf of other devices. DL queue in BBR is an internal device matter for graphs that originate or terminate in the device's DL			
ClockStability (ppm)	100	10	10	ClockStability values, in parts per million, are maximum allowed values over any continuous 30 s interval under industrial operating conditions. While low-cost I/O devices may have 100 ppm clocks, I/O devices in general should have a 25 ppm accuracy. This standard was designed assuming 25 ppm clocks in I/O devices, and it is anticipated that most application profiles will be constrained accordingly			
DLRoles	0000xxx1	0000xx1x	0000x11x	Bits indicate all of the DL roles that are supported by the device. Note that BBR is required to act as a router, such as for peer-to-peer messaging within a subnet			
AdvRate	0 (X)	6	6	All devices serving router and backbone router roles shall have sufficient resources to transmit an advertisement every 10 s (6 DPDUs per minute), on average. See 9.1.17			
ListenRate	0 (X)	36	36	All devices serving router and backbone roles shall have sufficient resources to operate their receivers for 36 seconds per hour (1%), on average. A mains powered BBR will normally be capable of running its receiver continuously, but some BBR classes (such as wireless bridges) might be energy constrained.			
TransmitRate	0 (X)	30	60	All devices serving router and backbone roles shall have sufficient resources to transmit the specified number of DSDUs per minute. See 9.1.17			

Table 486 describes baseline role profiles for the dlmo.Ch attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 486 – Role profiles: dlmo.Ch (channel hopping)**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	10	10	10	Five default hop sequences, numbered 1-5, are defined by the standard. A device can be provisioned or configured with up to 5 additional hop sequences			
MaxRowID (metadata)	127	127	127	One octet			

Table 487 describes baseline role profiles for the dlmo.TsTemplate attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 487 – Role profiles: dlmo.TsTemplate**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	8	10	10	Three default timeslot templates, numbered 1-3, are defined by the standard. These are included in the capacity			
MaxRowID (metadata)	127	127	127	One octet			

Table 488 describes baseline role profiles for the dlmo.Neighbor attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 488 – Role profiles: dlmo.Neighbor**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	2	8	32	An I/O shall support at least two neighbors, so that it can maintain two active DL routes for reporting. A router adds additional capacity to support routing on behalf of neighbors			
MaxRowID (metadata)	$2^{15}$	$2^{15}$	$2^{15}$	6LoWPAN unicast address limited to $2^{15}$			
GroupCode	O	M	M	GroupCode enables links to be used for multiple neighbors,			
ExtendGraph	O	O	O	Automatic extension of graphs is required for all devices. Support for the ExtendGraph field is an optional feature that provides a finer degree of control over graph extensions			

Table 489 describes baseline role profiles for the dlmo.Diagnostic attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 489 – Role profiles: dlmo.NeighborDiag**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	2*15 +9	3*15 +2*9	3*15 +2*9	Diagnostic capacity (metadata) is measured in octets.  Summary diagnostics, in Table 188 involve 15 octets of storage in the worst case. Actual storage and transmission may be more compact. Summary diagnostics are intended to be maintained on “publication” side of a given link, to collect diagnostics from the direction where more traffic flows. Summary diagnostics include a baseline clock diagnostic (ClockSigma).  More detailed clock diagnostics (Table 190) involve 9 octets of storage in the worst case. A summary clock diagnostic is provided along with the general diagnostic. Capacity is provided to collect these detailed clock diagnostics on an as-needed basis.			
MaxRowID (metadata)	2 <sup>15</sup>	2 <sup>15</sup>	2 <sup>15</sup>	6LoWPAN unicast address limited to 2 <sup>15</sup>			

Table 490 describes baseline role profiles for the dlmo.Superframe attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 490 – Role profiles: dlmo.Superframe**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	3	5	10	Default superframes for discovery of provisioning device are included in this count			
MaxRowID (metadata)	127	127	127	One octet			
AlwaysHop	O	O	O	Support for this feature is optional			

Table 491 describes baseline role profiles for the dlmo.Graph attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 491 – Role profiles: dlmo.Graph**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	2	8	16				
MaxRowID (metadata)	127	127	127	One octet			

Table 492 describes baseline role profiles for the dlmo.Link attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 492 – Role profiles: dlmo.Link**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	9	15	30	Default links for discovery of provisioning device are included in this count			
MaxRowID (metadata)	127	127	127	One octet			
Discovery	0,3	0,1,2,3	0,1,2	Discovery refers to bits 3/2 in Table 182. A system manager may be configured to discover routing-capable neighbors through active or passive scanning for advertisements			
JoinResponse	O	M	M				
NeighborType=2	O	M	M	Support of neighbor groups is mandatory for routing devices			

Table 493 describes baseline role profiles for the dlmo.Route attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 493 – Role profiles: dlmo.Route**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	3	1	64	I/O device has capacity for routing to the system manager, a first device, and a second device. Router needs only a route to the system manager. BBR needs at least one route (outbound route lookup) for each device in its sphere of influence, even if those routes are identical to each other.			
MaxRowID (metadata)	127	127	127	One octet			

Table 494 describes baseline role profiles for the dlmo.Queue\_Priority attribute. Those device elements not mentioned in this annex shall be supported as described in Clause 9.

**Table 494 – Role profiles: dlmo.Queue\_Priority**

Element	Status			Comments	Support		
	I/O	Router	BBR		N/A	Yes	No
Capacity (metadata)	O	2	2				
MaxRowID (metadata)	127	127	127	One octet			

## B.7 Network layer

Table 495 describes role profiles for routing table sizes.

**Table 495 – Routing table size**

Item number	Role types affected	Minimum # entries supported	Comments	Reference	Status	Support		
						N/A	Yes	No
RTS1	I/O	0			M			
RTS2	Router	0			M			
RTS3	Backbone router	15			M			

Table 496 describes role profiles for address table sizes.

**Table 496 – Address table size**

Item number	Role types affected	Minimum # entries supported	Comments	Reference	Status	Support		
						N/A	Yes	No
ATS1	I/O	4			M			
ATS2	Router	3			M			
ATS3	Backbone router	15			M			

## B.8 Transport layer

Table 497 describes role profiles for port support sizes.

**Table 497 – Port support size**

Item number	Role types affected	Minimum # entries supported	Comments	Reference	Status	Support		
						N/A	Yes	No
PSS1	I/O	2			M			
PSS2	Router	1			M			
PSS3	Backbone router	1			M			

## B.9 Application layer

Table 498 describes the minimum number of APs per role.

**Table 498 – APs**

Item number	Role types affected	Minimum # APs supported	Comments	Reference	Status	Support		
						N/A	Yes	No
UAP01	I/O	2		Clause 6,12.17	M			
UAP02	Router	1		Clause 6	M			
UAP03	Backbone router	1		Clause 6	M			
UAP04	Gateway	2		Clause 6, Clause 13	M			

NOTE The maximum number of contained objects supported includes the UAPMO.

## B.10 Gateway

Table 499 provides the role profile for a gateway.

**Table 499 – Role profile: Gateway**

Item number	Feature	Reference	Status	Comments	Support		
					N/A	Yes	No
GWRP1	Native access	13.3.1.5	O.1	Allows native service access only			
GWRP2	Interoperable tunnel mechanism	13.3.1.5	O.1	Allows tunneled access only			

Table 500 provides the role profile for a gateway native access.

**Table 500 – Role profile: Gateway native access**

Item number	Feature	Reference	Status	Comments	Support		
					N/A	Yes	No
GWRP1.1	Min IFOs supported	13.3.1.5	1				
GWRP1.2	Buffered message behavior	13.3.4		Constant, static, dynamic, non-cacheable.			
GWRP1.3	Min devices	Table 426	NSD	NSD >= 5			
GWRP1.4	Min leases	Table 426	2 x NSD - 3	NSD >= 5			

Table 501 provides the role profile for a gateway interoperable tunnel mechanism, where:

- FNG = number of foreign nodes behind gateway
- FNA = number of foreign nodes behind adapter(s)
- A = number of adapters

**Table 501 – Role profile: Gateway interoperable tunnel mechanism**

Item number	Feature	Reference	Status	Comments	Support		
					N/A	Yes	No
GWRP2.1	Min TUNs supported	13.3.1.5	GD x AD + 1	GD >= 1 AD >= 5			
GWRP2.1.1	Supports a foreign protocol	13.3.1.5	Annex O				
GWRP2.1.2	2-part tunneling	13.3.1.5					
GWRP2.1.3	TUN objects with Array of Tunnel endpoints attributes with multiple address elements	13.3.1.5	1				
GWRP2.1.3.1	# of elements in TUN with multiple address elements	13.3.1.5	A	A >= 5			
GWRP2.2	Min devices	Table 426	A	A >= 5			
GWRP2.3	Min leases	Table 426	2 x A	A >= 5			

## B.11 Provisioning

Table 502 provides the role profile devices implementing the I/O, router, gateway, or backbone router roles, all devices with a Type A field medium.

**Table 502 – Role profiles: I/O, routers, gateways, and backbone routers**

Item number	Feature	Reference	Status	Range	Comments	Support		
						N/A	Yes	No
DBPR-1	Joining a provisioning network using Key_Global = ISA100	14.6	M					
DBPR-2	Joining a provisioning network using Key_Join = OPEN	14.6	O		Default value of Key_Join = OPEN. Disabled once S is overwritten. Enabled again only if device reset to factory defaults			

## Annex C (INFORMATIVE)

### Background information

#### C.1 Industrial needs

The wireless needs for industrial applications are significantly different than those required for residential, commercial, or military applications. These differences stem from the unique industrial ranking of priorities of characteristics such as device cost, system cost, lifecycle cost, reliability, maintainability, consistency, robustness, extensibility, security, coexistence, regulatory restrictions, and interoperability.

ISA100 committee members collected and analyzed more than 500 use cases to define more completely the wireless communication needs of the industrial sector. The major conclusions of this effort were:

- Opportunity: Non-existent wireless sensing is an opportunity for end users, vendors, and emerging standards.
- Interoperable: Since multiple vendor instrument facilities dominate the industrial environment, wireless standards should be of high value.
- Integration: Multiple communication paths between devices are needed, especially to distributed control system (DCS) instruments.
- Applications: Applications such as monitoring/alerting are of greatest immediate interest since they constitute the largest potential use of wireless devices.
- Reliability and security: Critical factors for emerging standards and vendors.
- Power: Battery life expectations will vary due to application, environment, cost constraints, etc. Some devices will have mains power, while others will be powered by batteries or will scavenge energy from the environment.

#### C.2 Usage classes

##### C.2.1 General

While there are many techniques that may be used to categorize the communications needs of industrial applications, this standard uses classes based upon usage. Analysis of the patterns of intended use of inter-device industrial wireless communications resulted in a partitioning of such communications into six classes. These classes are summarized in Table 503.

**Table 503 – Usage classes**

<b>Safety</b>	Class 0: Emergency action	Always critical	Importance of message timeliness increases → <small>*Batch levels as defined by ISA S88; where L3 = unit and L4 = process cell</small>
<b>Control</b>	Class 1: Closed loop regulatory control	Often critical	
	Class 2: Closed loop supervisory control	Usually non-critical	
	Class 3: Open loop control	Human in the loop	
		NOTE Batch levels* 3 & 4 could be class 2, class 1 or even class 0, depending on function	
<b>Monitoring</b>	Class 4: Alerting	Short-term operational consequence (e.g., event-based maintenance)	
	Class 5: Logging and downloading / uploading	No immediate operational consequence (e.g., history collection, sequence-of-events, preventive maintenance)	

### C.2.2 Class examples

#### C.2.2.1 Class 0: Emergency action (always critical)

Class 0 examples include:

- Safety interlock
- Emergency shutdown
- Automatic fire control

#### C.2.2.2 Class 1: Closed loop regulatory control (often critical)

Class 1 examples include:

- Direct control of primary actuators (e.g., field device to host connection availability on demand of 99.99% or more, with link outages > 500 ms intolerable, with demand rates typically once every 4 s)
- High frequency cascades

#### C.2.2.3 Class 2: Closed loop supervisory control (usually non-critical)

Class 2 examples include:

- Low frequency cascade loops
- Multivariable controls
- Optimizers

#### C.2.2.4 Class 3: Open loop control (human in the loop)

Class 3 examples include:

- Operator manually initiates a flare and watches the flare
- Guard remotely opens a security gate
- Operator performs manual pump/valve adjustment

#### C.2.2.5 Class 4: Alerting – Short-term operational consequence

Class 4 examples include:

- Event-based maintenance
- Marginal bearing temp results in technician sent to field
- Battery low indicator for a device results in technician sent to change battery
- Asset tracking

#### C.2.2.6 Class 5: Logging – data/messages with no immediate operational consequence

Class 5 examples include:

- History collection
- Preventive maintenance rounds
- Sequence of events (SOE) uploading

NOTE SOE requires lossless communication, such as file transfer, rather than timely communication such as required by control messaging.

### C.3 Other uploading and downloading- alarms (human or automated action)

Alarm examples include:

- Class 0: leak detector for radiation or fatally toxic gas, automated response (e.g., automated containment response)
- Class 1: high-impact process condition, automated response (e.g., automated shutdown of reaction)

- Class 2: automated response to process condition (e.g., automated flow diversion)
- Class 3: process condition with manually-initiated operational response (e.g., decide whether to divert flow to a parallel reactor)
- Class 4: equipment condition with short-time-scale maintenance response (e.g., send technician to field)
- Class 5: equipment condition with long-time-scale maintenance action (e.g., order spare parts)

## C.4 The open systems interconnection basic reference model

### C.4.1 Overview

This standard defines the protocol suite of the wireless network. A protocol suite is a particular software implementation of a networking protocol suite. In practical implementation, protocol suites are often divided into layers such as those defined by the Open Systems Interconnection-Basic Reference Model ISO/IEC 7498-1:1994(E). The format in this standard is based upon this reference model (see Figure 169), implementing five of the basic reference models seven layers.

**NOTE** It is important to note that this is a virtual model, which does not require implementations, or even specifications, to be partitioned along the same lines.

OSI layer	Function	ISA100.11a
Application	Provides the user with network-capable application	<ul style="list-style-type: none"> <li>• Uses object-oriented approach to encapsulate data and functionality</li> <li>• Maps legacy protocols within the constraints of ISA100.11a</li> <li>• Ensures an open and interoperable application environment</li> <li>• Provides a common integration point to multiple host control systems secured session between network devices</li> </ul>
Presentation	Converts application data between network and local machine formats	
Session	Provides connection management services for applications	
Transport	Enables network-independent, transparent message transfer	Provides connectionless services (based upon UDP) with optional security
Network	Provides end-to-end routing of packets; resolving network addresses	Provides network addressing, address translation, fragmentation and reassembly, and network routing
Data link	Establishes data packet structure, framing, error detection, bus arbitration	Provides secure, robust, and reliable links, time synchronization for time division multiple access and channel hopping
Physical	Provides mechanical/electrical connection; transmits raw bitstream	Uses 2.4 GHz band, IEEE 802.15.4 radios

**Figure 169 – Basic reference model**

The upper layer, application (AL), of the basic reference model of this standard provides local functionality for one or more associated user application processes (UAPs).

The four lower layers, transport (TL), network (NL), data link (DL), and physical (PhL), are devoted to data communication. Each has the capability of multiplexing and demultiplexing, and of splitting and merging information flows from adjacent layers. In other words, the messaging relationships between an AL entity and a TL entity, or between a TL entity and an NL entity, or between an NL entity and a DL entity, or between a DL entity and a PhL entity, do not have to be one-to-one.

These lower layers also have the following abilities:

- To sequence service data units (SDUs) to maintain the order of original presentation;
- To segment (or reassemble) and block (or deblock) SDUs into protocol data units (PDUs) and to concatenate (or separate) PDUs so that they are sized more appropriately for the conveyance capabilities of the lower layer;
- To split (or recombine) PDUs among multiple lower layer routes; and

- To acknowledge receipt of PDUs as a form of error control.

#### C.4.2 Application layer (AL)

The AL is the layer that interfaces directly to (and conceptually includes) UAPs, managing communications with other UAPs under the guidance of the local management UAP. A UAP may perform an individual function or any combination of functions. UAPs may be used, for example, to:

- Handle input and/or output hardware;
- Distribute communications to a set of co-resident UAPs within a device (proxy function);
- Support tunneling of a non-native (e.g., control system legacy) protocol compatible with the network environment described in this standard; and/or
- Perform a computational function.

The AL is typically composed of one or more UAPs that share common service elements.

The primary tasks of an AL entity are:

- To provide a place in the architecture of this standard for UAPs;
- To provide the means by which UAPs manage communications with user application processes for other devices through the protocol suite, including:
- Identification of intended communications partners (e.g., by name, by address, by description, etc.)
- Agreement on security aspects (e.g., authentication, data integrity)
- Determination of acceptable quality of service (e.g., priority, time windows for control messaging, acceptability of out-of-order message delivery, acceptability of message delivery in partial increments, etc.)
- Agreement on responsibility for error recovery
- Identification of abstract syntaxes
- Synchronization of cooperating UAPs
- To provide the means by which UAPs can inform the associated application entity of needed resource requirements, including those for message buffering:
- Expected and maximum message sizes
- Maximum expected burstiness of message transmission and reception or how many messages can be sent or arrive within a short amount of time as compared to the average periodicity of messages
- To provide any necessary communication functions that are not already performed by the lower layers.

#### C.4.3 Transport layer

The TL is the highest layer at which communicating applications are addressable. The primary tasks of a TL entity are:

- To provide addressing of user application processes via selection of a specific associated AL entity;
- To establish end-to-end messaging paths from one user application process to one or more other user application processes via their associated AL entities, where those processes are usually in separate devices;
- To convey and regulate the flow of messages between or among those user application processes; and
- To terminate those messaging paths when appropriate.

#### C.4.4 Network layer

The NL is the highest layer at which communicating devices are addressable. It is the lowest layer with more than local scope, which forwards messages between one entity group and others, or discards the messages. The primary tasks of an NL entity are:

- To provide network-wide addressing of devices;
- To relay messages (NPDUs) between entities (e.g., a router) via DL subnets, usually changing source and destination DL entity addresses associated with the message envelopes (DPDUs) in the process, or to discard the NPDUs; and
- To provide segmentation and reassembly of messages, as appropriate, to match the capabilities of the DL subnets on which messages are being forwarded.

NOTE The network layer is the OSI layer where endpoint device addressing and routing occurs. Lower layer relay can forward messages within a single addressing domain without message modification, but cannot readdress messages or span addressing domains. Network-wide device addresses are N addresses.

#### C.4.5 Data link layer

The DL is the lowest information-centric layer, which coordinates interacting PhL entities and provides basic low-level messaging among DL entities. The primary tasks of a DL entity are:

- To provide link-local addressing of peer-DL entities;
- To convey messages (DPDUs) from one DL entity to all others whose PhL entities are correspondents (e.g., to all PhL entities of the local link), or to discard the DPDUs;
- To manage use of the PhL;
- To provide low-level message addressing, message timing and message integrity checks;
- To provide low-level detection of and recovery from message loss (e.g., immediate acknowledgement; retry if no acknowledgement); and
- Optionally, to relay DPDUs between DL entities (e.g., a bridge).

NOTE The DL is the OSI layer that manages and compensates for the specific characteristics of the selected physical communications technology. It provides only local addressing; it can forward messages within the local addressing domain without readdressing, but cannot modify message addresses. DL addresses have only local scope, which may be duplicated in other local links.

#### C.4.6 Physical layer

The PhL is the lowest layer of the OSI model and the only layer that deals with real-world physics. All other layers deal with abstract information, ultimately represented as bits; the PhL is concerned with physical signals (sometimes referred to as baud). The primary tasks of a PhL entity are:

- To code bits, either singly or in multi-bit groups, into physical signals;
- To convey those signals from one physical location to another;
- To decode those signals into single-bit or multi-bit groups, possibly with error correction;
- To take direction from the associated DL entity with respect to physical channel setup, physical receiver addressing and other aspects of the communications channel and coding;
- To convey to the locally-associated DL entity information about the state of the PhL entity, the channel and the last set of received signals; and
- Optionally, to relay PhL PDUs between PhL entities (e.g., a repeater).

## Annex D (NORMATIVE)

### Configuration defaults

#### D.1 General

This annex summarizes the default settings for configuration.

#### D.2 System management

Table 504 lists the system management configuration defaults.

**Table 504 – System management configuration defaults**

Name	Initial default value	Reference
Confirmation_Timeout_Device_Diagnostics	10	Table 7
Alerts_Disable_Device_Diagnostics	0	Table 7
Confirmation_Timeout_Comm_Diagnostics	10	Table 7
Alerts_Disable_Comm_Diagnostics	0	Table 7
Confirmation_Timeout_Security	10	Table 7
Alerts_Disable_Security	0	Table 7
Confirmation_Timeout_Process	10	Table 7
Alerts_Disable_Process	0	Table 7
Comm_Diagnostics_Alarm_Recovery_AlertDescriptor	Alert report disabled = False Alert report priority = 3	Table 7
Security_Alarm_Recovery_AlertDescriptor	Alert report disabled = False Alert report priority = 3	Table 7
Device_Diagnostics_Alarm_Recovery_AlertDescriptor	Alert report disabled = False Alert report priority = 3	Table 7
Process_Alarm_Recovery_AlertDescriptor	Alert report disabled = False Alert report priority = 3	Table 7
DL_Alias_16_Bit	0	Table 10
Network_Address_128_Bit	0	Table 10
Device_Power_Status_Check_AlertDescriptor	Alert report disabled = False Alert report priority = 8	Table 10
DMAP_State	1	Table 10
Join_Command	0	Table 10
Static_Revision_Level	0	Table 10
Restart_Count	0	Table 10
Uptime	0	Table 10
TAI_Time	0	Table 10
Comm_SW_Major_Version	0	Table 10
Comm_SW_Minor_Version	0	Table 10
System_Manager_128_Bit_Address	0	Table 10
System_Manager_EUI64	0	Table 10
System_Manager_DL_Alias_16_Bit	0	Table 10
Contract_Request_Timeout	30	Table 10
Max_ClientServer_Retries	3	Table 10
Max_Retry_Timeout_Interval	30	Table 10
DMAP_Objects_Count	1	Table 10
Warm_Restart_Attempts_Timeout	60	Table 10
Current_UTC_Adjustment	0	Table 25
Next_UTC_Adjustment_Time	See Table 25	Table 25
Next_UTC_Adjustment	34	Table 25

### D.3 Security

Table 505 lists the security configuration defaults.

**Table 505 – Security configuration defaults**

Name	Initial default value	Reference
Security_Level	1	Table 88
Protocol_Version	1	Table 93
DL_Security_Level	1	Table 93
Transport_Security_Level	1	Table 93
Join_Timeout	60 s	Table 93
MPDU_MIC_Failure_Limit	5	Table 93
MPDU_MIC_Failure_Time_Unit	60 s	Table 93
TPDU_MIC_Failure_Limit	5	Table 93
TPDU_MIC_Failure_Time_Unit	5	Table 93
DSMO_KEY_Failure_Limit	1	Table 93
DSMO_KEY_Failure_Time_Unit	1	Table 93
Security_MPDU_Fail_Rate_Exceeded_AlertDescriptor	Alert Report disabled = False, Alert Report Priority = 6	Table 93
Security_TPDU_Fail_Rate_Exceeded_AlertDescriptor	Alert Report disabled = False, Alert Report Priority = 6	Table 93
Security_Key_Update_Fail_Rate_Exceeded_AlertDescriptor	Alert Report disabled = False, Alert Report Priority = 6	Table 93
pduMaxAge	510	Table 93
SoftLifeTime	50	Table 94
DSMO alert type 0 = Security_MPDU_Fail_Rate_Exceeded	0	Table 98
DSMO alert type 1 = Security_TPDU_Fail_Rate_Exceeded	0	Table 98
DSMO alert type 2 = Security_Key_Update_Fail_Rate_Exceeded	0	Table 98

### D.4 Data link layer

Table 506 lists the DL configuration defaults.

**Table 506 – DL configuration defaults**

Name	Initial default value	Reference
ActScanHostFract	0	Table 141
AdvJoinInfo	Null	Table 141
AdvSuperframe	0	Table 141
SubnetID	0	Table 141
SolicTemplate	Null	Table 141
AdvFilter	See 9.4.2.20	Table 141
SolicFilter	See 9.4.2.20	Table 141
TaiAdjust	Null	Table 141
MaxBackoffExp	5	Table 141
MaxDsduSize	96	Table 141
MaxLifetime	120 (30 s)	Table 141
NackBackoffDur	60 (15 s)	Table 141
LinkPriorityXmit	8	Table 141
LinkPriorityRcv	0	Table 141
EnergyDesign	See 9.4.2.22	Table 141
DeviceCapability	See 9.4.2.23	Table 141
IdleChannels	0	Table 141
ClockExpire	See 9.4.2.1	Table 141
ClockStale	45	Table 141
RadioSilence	600	Table 141
RadioSleep	0	Table 141
RadioTransmitPower	See 9.4.2.1	Table 141
CountryCode	0x0800	Table 141
Candidates	Null	Table 141
DiscoveryAlert	60	Table 141
SmoothFactors	See Table 153	Table 141
QueuePriority	N=0	Table 141
Ch	See 9.4.3.2	Table 141
TsTemplate	See 9.4.3.3	Table 141
Neighbor	Empty	Table 141
Superframe	Empty	Table 141
Graph	Empty	Table 141
Link	Empty	Table 141
Route	Empty	Table 141
NeighborDiag	Empty	Table 141
ChannelDiag	See 9.4.2.27	Table 141
Receive template parameters	See Table 165	Table 165
Transmit template parameters	See Table 166	Table 166
Receive template for scanning parameters	See Table 167	Table 167

## D.5 Network layer

Table 507 lists the network layer configuration defaults.

**Table 507 – Network configuration defaults**

Name	Initial default value	Reference
Enable_Default_Route	Disabled	Table 206
Max_NSDU_size	70	Table 206
Frag_Reassembly_Timeout	60	Table 206
Frag_Datagram_Tag	Random	Table 206
DroppedNPDUAlertDescriptor	Alert report disabled = True Alert report priority = 7	Table 206
Source_Address*	0	Table 207
Destination_Address	0	Table 207
Contract_Priority	00	Table 207
Include_Contract_Flag	0	Table 207
NWK_HopLimit	64	Table 208
Outgoing_Interface	0	Table 208

## D.6 Transport layer

Table 508 lists the transport layer configuration defaults.

**Table 508 – Transport configuration defaults**

Name	Initial default value	Reference
MaxNbOfPorts	15	Table 229
TPDUin	0	Table 229
TPDUinRejected	0	Table 229
TSDUout	0	Table 229
TSDUin	0	Table 229
TSDUinRejected	0	Table 229
TPDUout	0	Table 229
IllegalUseOfPortAlertDescriptor	Alert report disabled = True Alert report priority = 8 (medium)	Table 229
TPDUonUnregisteredPortAlertDescriptor	Initial default value: Alert report disabled = True Alert report priority = 4 (low)	Table 229
TPDUoutOfSecurityPoliciesAlertDescriptor	Initial default value: Alert report disabled = True Alert report priority = 2 (journal)	Table 229

## D.7 Application layer

Table 509 lists the application layer configuration defaults.

**Table 509 – Application configuration defaults**

Name	Initial default value	Reference
ObjectIdentifier	0	Table 240
UAP_ID	0=N/A	Table 240
UAP_TL_Port	0=N/A	Table 240
State	Active	Table 240
Command	0=None	Table 240
MaxRetries	3	Table 240
Number of unscheduled communication correspondents	0=N/A	Table 240
Number of objects in the UAP including this UAPMO	1	Table 240
Static_Revision_Level	0	Table 240
Categories	0	Table 243
Errors	0	Table 243
State	0=Idle	Table 246
MaxBlockSize	1 to (MaxNPDUsize + Max TL header size – max(sizeof (additional coding of Application layer UploadData service request), additional coding of sizeof(Application layer DownloadData service response)))	Table 246
MaxDownloadSize	0	Table 246
MaxUploadSize	0	Table 246
DownloadPrepTime	0	Table 246
DownloadActivationTime	0	Table 246
UploadPrepTime	0	Table 246
UploadProcessingTime	0	Table 246
DownloadProcessingTime	0	Table 246
CutoverTime	0	Table 246
LastBlockDownloaded	0	Table 246
LastBlockUploaded	0	Table 246
ErrorCode	0 =(noError)	Table 246
Revision	0	Table 256
CommunicationEndpoint	The configured connection endpoint valid element indicates not configured (i.e., endpoint is not valid)	Table 256
MaximumItemsPublishable	Local matter	Table 256
NumberItemsPublishing	0	Table 256
Array of ObjectAttributeIndexAndSize	Element size is 0	Table 256
Concentrator ContentRevision	0	Table 258
CommunicationEndpoint	The configured connection endpoint valid element indicates not configured (i.e., endpoint is not valid)	Table 258
MaximumItemsSubscribing	Local matter	Table 258
NumItemsSubscribing	0	Table 258
Array of ObjectAttributeIndexAndSize	Element size is 0	Table 258
Protocol	Local matter (protocol-specific)	Table 260
Status (Configuration status)	0	Table 260
Period (Data publication period)	0	Table 260
Max_Peer_Tunnels	0	Table 260
Num_Peer_Tunnels	0	Table 260
ObjectIdentifier	7	Table 283
MalformedAPDUsAdvise	Disabled	Table 283
TimeIntervalForCountingMalformedAPDUs	0	Table 283
MalformedAPDUsThreshold	0	Table 283
MalformedAPDUAAlertDescriptor	Alert report disabled = True Alert report priority = 7	Table 283
PV	NaN	Table 287
Mode	Actual mode value indicates OOS	Table 287

Scale	Engineering units values for 0% and for 100% BOTH indicate 0	Table 287
OP	NaN	Table 290
Mode	Actual mode value indicates OOS	Table 290
Readback	NaN	Table 290
Scale	Engineering units values for 0% and for 100% BOTH indicate 0	Table 290
PV_B	0	Table 293
Mode	Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access	Table 293
OP_B	0	Table 296
Mode	Read only for actual mode; target mode, permitted mode, and normal mode all have read/write access	Table 296
Readback_B	0	Table 296
Value	See IEEE 754 specification	Table 300
Target	OOS	Table 302
Actual	OOS	Table 302
Permitted	OOS	Table 302
Normal	OOS	Table 302
Engineering units at 100%	0	Table 304
Engineering units at 0%	0	Table 304
Decimal point location	0	Table 304

## D.8 Gateway

Table 510 lists the gateway configuration defaults.

**Table 510 – Gateway configuration defaults**

Name	Initial default value	Reference
Max_Devices	0	Table 419

## D.9 Provisioning

Table 511 lists the provisioning configuration defaults.

**Table 511 – Provisioning configuration defaults**

Name	Initial default value	Reference
Default_Nwk_ID	1	Table 421
Default_SYM_Join_Key	ISA100	Table 421
Open_SYM_Join_Key	OPEN	Table 421
Default_Frequency_List	7FFF	Table 421
Join_Method_Capability	00	Table 421
Allow_Provisioning	Allowed (1)	Table 421
Allow_Over_The_Air_Provisioning	Allowed (1)	Table 421
Allow_OOB_Provisioning	Allowed (1)	Table 421
Allow_Reset_to_Factory_defaults	Allowed (1)	Table 421
Allow_Default_Join	Allowed (1)	Table 421
Target_Nwk_ID	0	Table 421
Target_Nwk_BitMask	1111111111111111	Table 421
Target_Join_Method	1	Table 421
Number of PKI_Certificates	1	Table 421
Current_UTC_Adjustment	34	Table 421
White_List	[]	Table 424
SYM_Key_List	[ISA100]	Table 424
SYM_Key_Expiry_Times	FFFFFF (Infinity)	Table 424
Target_NWK_ID	0	Table 424
Target_Join_Method	1	Table 424
Target_Join_Time	0	Table 424
Allow_Provisioning	Allowed (1)	Table 424
Allow_Default_Join	Allowed (1)	Table 424
Device_Specific_Provisioning_Flag	0=disabled	Table 424
DPSO_Alerts_AlertDescriptor	Alert report disabled = False Alert report priority = 6	Table 424
Current_UTC_Adjustment	34	Table 424
Device_EUI	1	Table 425
Device_SYM_Key	ISA100	Table 425
SYM_Key_Expiry_Time	FFFFF (Infinity)	Table 425
Target_Nwk_ID	0	Table 425
Target_Join_Method	1	Table 425
Allow_Provisioning	1	Table 425
Allow_Default_Join	1	Table 425

## **Annex E (INFORMATIVE)**

### **Use of backbone networks**

#### **E.1 General**

Use of a backbone network can be advantageous to the system designer, since it takes the message off of the Type A field medium, allowing additional bandwidth and higher QoS for other messages.

#### **E.2 Recommended characteristics**

Although the backbone itself is not specified within this standard, it is assumed and recommended that the backbone will have the following characteristics:

- Throughput equal to or greater than the throughput of the Type A field medium ( $\geq 250$  kbit/s).
- Capability of supporting two-way unsolicited message traffic.
- Quality of service of a sufficient level such that time synchronization can be maintained across the network. This may place specific time synchronization methods on the backbone.
- High reliability. Operation should not burden the network with frequent lost messages and retries.
- Security sufficient not to present a security threat to the end users application or the network.
- Capability of either encapsulating (tunneling) protocol TPDUs or TSDUs defined by this standard or translating them such that they may traverse the backbone without being modified when emerging at the backbone devices. In general, the backbone shall be able to take a standard-compliant TSDU from the point of ingress and deliver it across the backbone to the point of egress unmodified.
- Capability of preserving the end-to-end application security mechanisms.
- Support for multipoint networking between devices.

It is recognized that many standard fieldbuses may not have these characteristics and therefore may not be suitable for use as a backbone network. In many cases, a backbone network will be an IP network such as Ethernet IEEE 802.3 or wireless IEEE 802.11, but there is no requirement for this. There are many other alternatives in the marketplace that exist and are well-suited for the purposes of a backbone network. These might include simple point-to-point or point-to-multipoint wireless networks.

#### **E.3 Internet protocol backbones**

##### **E.3.1 Methods of IPv6 protocol data unit transmission**

###### **E.3.1.1 General**

In many cases, an available backbone will use an Internet Protocol (IP) network layer. In this case there are many different ways to transport the wireless industrial sensor network (WISN) TPDUs using standardized protocol behavior:

###### **E.3.1.2 Encapsulate wireless industrial sensor network transport protocol data units within IPv4 frames**

The mechanism used to encapsulate WISN TPDUs within IPv4 frames is formally known as IPv6 over IPv4 or 6over4 and is sometimes called virtual Ethernet. This method is documented in IETF RFC 2529 (see <http://www.ietf.org/rfc/rfc2529.txt>). A backbone router (IETF RFC 2529 refers to them as IPv6 hosts) located on a physical link that has no directly

connected IPv6 router may become a fully functional IPv6 host by using an IPv4 multicast domain as its virtual local link. Backbone routers connected using this method do not require IPv4-compatible addresses or configured tunnels

### **E.3.1.3 Tunnel wireless industrial sensor network transport protocol data units over IPv4 network**

Following IETF RFC 1933 (<http://www.ietf.org/rfc/rfc1933.txt>), this method encapsulates IPv6 protocol data units (PDUs) within IPv4 headers to carry them over IPv4 routing infrastructures. Two types of tunneling are possible, configured and automatic. In configured tunneling, the IPv4 tunnel endpoint address is determined by configuration information on the encapsulating node. In automatic tunneling, the IPv4 tunnel endpoint address is determined from the IPv4 address embedded in the IPv4-compatible destination address of the IPv6 PDU.

### **E.3.1.4 Encapsulate wireless industrial sensor network transport protocol data units within raw Ethernet frames**

This method specifies the frame format for transmission of IPv6 PDUs following IETF RFC 2464 (<http://www.ietf.org/rfc/rfc2464.txt>). Furthermore, this method dictates the formation of IPv6 link-local addresses and statelessly autoconfigured addresses on Ethernet networks. Finally, this approach specifies the content of the source/target link-layer address option used in router solicitation, router advertisement, neighbor solicitation, neighbor advertisement, and redirect messages when those messages are transmitted on an Ethernet network.

### **E.3.1.5 Use native IPv6 backbone without any encapsulation or tunneling**

If the backbone uses an IPv6 network layer, neither encapsulation nor tunneling is necessary, since the backbone native mode is to transport IPv6 PDUs.

## **E.3.2 Backbone router peer device discovery**

For the backbone router (BBR) to function properly and to connect WISN devices on the backbone, it needs to know the backbone addresses of the other BBRs or peers in the backbone network. Within each BBR, the addressing information of its peers should be stored in a backbone router peer table (BRPT). There are two basic methods of generating the BRPT, configuration and discovery.

NOTE The BRPT and the mechanism for discovering peers are beyond the scope of this standard.

Configuration occurs when the addresses of peer BBRs are inserted into the BRPT by the system manager or an operator. The advantages of this method are that it is straightforward and prevents the BBR from accessing inappropriate devices on the backbone.

Discovery occurs when the BBR automatically searches the backbone for peer devices. There are multiple discovery techniques, such as those mentioned in IETF RFC 2529, UPNP, and others. The advantages of this method are that it is automatic, requires no operator involvement, and can be easily and often updated.

## **E.3.3 Security**

### **E.3.3.1 Security of transport protocol data units**

The security mechanisms of the backbone are beyond the scope of this standard. Typical IP security methods include IPSec, SSL, and others. In addition to any security mechanisms on the backbone, the WISN TL security mechanism protects the TPDU within the backbone.

### **E.3.3.2 Security of the backbone**

There is a perception by some that allowing a WISN to access an IP backbone could degrade the security of the backbone. This concern may be mitigated by restricting the BBR access to only peer BBRs via an access control list or by the use of firewalls set up to restrict access properly to specific devices.

## **Annex F (NORMATIVE)**

### **Basic security concepts – Notation and representation**

#### **F.1 Strings and string operations**

A string is a sequence of symbols over a specific set (e.g., the binary alphabet (0,1) or the set of all octets).

The length of a string is the number of symbols it contains (over the same alphabet).

The right-concatenation of two strings  $x$  and  $y$  of length  $m$  and  $n$  respectively (notation  $x \parallel y$ ) is the string  $z$  of length  $m+n$  that coincides with  $x$  on its leftmost  $m$  symbols and with  $y$  on its rightmost  $n$  symbols.

An octet is a symbol string of length 8. In this context, all octets are strings over the binary alphabet.

#### **F.2 Integers, octets, and their representation**

Throughout this specification, the representation of integers as octet strings and of octet strings as binary strings shall be fixed.

All integers shall be represented as octet strings in most-significant-octet-first order. This representation conforms to the convention in 4.3 of ANSI X9.63:2001.

All octets shall be represented as binary strings in most-significant-bit-first order.

#### **F.3 Entities**

Throughout this specification, each entity shall be a DEV and shall be uniquely identified by its 64-bit IEEE device address. The parameter entlen shall have the integer value 64.

## Annex G (INFORMATIVE)

### **Using certificate chains for over-the-air provisioning**

This standard uses implicit certificate called the PublicReconstrKey (see Annex H for details) for the asymmetric key-based cryptography. Given the identity of a device A ( $ID_A$ ) and the implicit certificate  $\gamma_A$  of the device, the asymmetric key of the device A can be computed using the following equation:

$$Q_A = \text{Hash}(\gamma_A || ID_A)\gamma_A + Q_{CA}$$

where  $Q_{CA}$  is the asymmetric key of the certificate authority (CA).

With this background, the following steps outline the process for OTA provisioning using asymmetric key cryptography as outlined in Figure 167.

- 1) The CA publishes  $Q_{CA}$ , its asymmetric key, on the web.
- 2) The device manufacturer (DM) gets a certificate from the CA:

$$C_{DM} = \text{PublicReconstrKey}(DM) || \text{Subject}(DM) || \text{Issuer}(CA) || \text{Text}$$

where:

- $\text{Subject} = ID$  of the DM
- $\text{Issuer} = ID$  of the CA
- $\text{PublicReconstrKey}_\text{DM} = \gamma_\text{DM}$  is used to calculate the asymmetric key of the DM using the equation:

$$Q_{DM} = \text{HASH}(\gamma_{DM} || \text{Subject})\gamma_{DM} + Q_{CA}$$

- 3) The individual device gets a certificate from the DM:

$$C_{DEV} = \text{PublicReconstrKey}(DEV) || \text{Subject}(DEV) || \text{Issuer}(DM) || \text{Text}$$

where:

- $\text{Subject} = ID$  of the device
- $\text{Issuer} = ID$  of the DM
- $\text{PublicReconstrKey}_\text{DEV} = \gamma_\text{DEV}$  is used to calculate the asymmetric key of the device using the equation:

$$Q_{DEV} = \text{HASH}(\gamma_{DEV} || \text{Subject})\gamma_{DEV} + Q_{DM}$$

- $C_{DEV}$  and  $C_{DM}$  are populated in the DBP by the DM.
- 4) The DBP joins the PD in a provisioning network. The PD has  $Q_{CA}$ .
  - 5) The DBP sends a random number,  $C_{DEV}$ , and  $C_{DM}$  to the PD. The PD calculates  $Q_{DEV}$  as explained in steps 2 and 3.
  - 6) A challenge/response mechanism is used to authenticate the device, and the security manager should validate the manufacturer's implicit certificate at this point.
  - 7) If the challenge/response is passed, the PD sends  $K_{join}$  encrypted with  $Q_{DEV}$ .

## Annex H (NORMATIVE)

### Security building blocks

#### **H.1 Symmetric key cryptographic building blocks**

##### **H.1.1 Overview**

The following symmetric key cryptographic primitives and data elements are defined for use with all security processing operations specified in this standard.

##### **H.1.2 Symmetric key domain parameters**

The symmetric key shall have key size keylen=128 (in bits).

##### **H.1.3 Block cipher**

The block cipher used in this specification shall be the Advanced Encryption Standard AES-128, as specified in FIPS 197. This block-cipher shall be used with symmetric keys as specified in H.1.2. Hence, the key size is equal to the block size of the block-cipher, 128 bits.

##### **H.1.4 Mode of operation**

The block-cipher mode of operation used in this specification shall be the CCM\* mode of operation, as specified in Annex B.3.2 of IEEE Std 802.15.4-2006.

##### **H.1.5 Cryptographic hash function**

The cryptographic hash function used in this specification shall be the block cipher-based cryptographic hash function specified in H.9, with the following instantiations:

- Each entity shall use the block-cipher E as specified in H.1.3;
- All integers and octets shall be represented as specified in F.2.

The Matyas-Meyer-Oseas hash function (see H.9) has a message digest size hashlen that is equal to the block size, in bits, of the established block cipher.

##### **H.1.6 Keyed hash function for message authentication**

The keyed hash message authentication code (HMAC) used in this specification shall be HMAC, as specified in the FIPS 198, with the following instantiations:

- Each entity shall use the cryptographic hash H function as specified in H.1.5;
- The block size B shall have the integer value B=keylen/8, where keylen is as specified in H.1.2 (i.e., B is equal to the length of the symmetric key, in octets, that is used by the keyed hash function).
- The output size HMAClen of the HMAC function shall have the same integer value as the message digest parameter hashlen, as specified in H.1.5.

##### **H.1.7 Specialized keyed hash function for message authentication**

The specialized<sup>11</sup> keyed hash message authentication code used in this specification shall be the keyed hash message authentication code, as specified in H.1.6.

---

<sup>11</sup> This refers to a MAC scheme where the MAC function has the additional property that it is also pre-image and collision resistant for parties knowing the key (see also Remark 9.8 of Menezes et al.). Such MAC functions allow key derivation in contexts where unilateral key control is undesirable.

### **H.1.8 Challenge domain parameters**

The challenge domain parameters used in the specification shall be as specified in H.6.2, with the instantiation ( $\text{minchallengelen}$ ,  $\text{maxchallengelen}$ )=( $\text{keylen}$ ,  $\text{keylen}$ ), where  $\text{keylen}$  is as specified in H.1.2.

All challenges shall be validated using the challenge validation primitive as specified in H.7.

## **H.2 Asymmetric key cryptographic building blocks**

### **H.2.1 General**

The following asymmetric key cryptographic primitives and data elements are defined for use with all security processing operations specified in this standard.

### **H.2.2 Elliptic curve domain parameters**

The elliptic curve domain parameters used in this specification shall be those for the curve  $\text{ansit283k1}$  as specified in Appendix J4.5, Example 1, of ANSI X9.63:2001.

All elliptic curve points shall be validated using the public key validation primitive as specified in 5.2.2 of ANSI X9.63:2001.

### **H.2.3 Elliptic curve point representation**

All elliptic curve points shall be represented in polynomial notation as specified in 4.1.2.1 of ANSI X9.63:2001. All elliptic curve points shall be transmitted in compressed form, as specified in 4.2.2 of ANSI X9.63:2001.

### **H.2.4 Elliptic curve public-key pair**

An elliptic curve-key pair consists of an integer  $q$  and a point  $Q$  on the curve determined by multiplying the generating point  $G$  of the curve by this integer (i.e.,  $Q=qG$ ) as specified in ANSI X9.63:2001. Here,  $Q$  is called the public key, whereas  $q$  is called the private key; the pair  $(q, Q)$  is called the public-key pair. Each private key shall be represented as specified in 4.3.1 of ANSI X9.63:2001. Each public key shall be on the curve as specified in H.2.2 and shall be represented as specified in H.2.3.

## **H.3 Keying information**

### **H.3.1 General**

The following specifies the format of asymmetric-key keying information used in this standard.

### **H.3.2 Elliptic curve cryptography implicit certificates**

Implicit certificates  $IC_U$  shall be specified as  $IC_U = \text{PublicKeyReconstrData} \parallel \text{Subject} \parallel \text{Issuer} \parallel \text{Usage_Serial} \parallel \text{KeyValidityInfo} \parallel \text{Text}$ , where:

- The parameter  $\text{PublicKeyReconstrData}$  shall be the public-key reconstruction data  $BEU$  as specified in the implicit certificate generation scheme (see H.5.1).
- The parameter  $\text{Subject}$  shall be the entity  $U$  that is bound to the public key reconstruction data  $BEU$  during execution of the implicit certificate generation scheme, i.e., the entity that purportedly owns the private key corresponding to the public key that can be reconstructed from  $\text{PublicKeyReconstrKey}$ .
- The parameter  $\text{Issuer}$  shall be the entity of the certificate authority (CA) that creates the implicit certificate during the execution of the implicit certificate generation scheme.
- The parameter  $\text{Usage_Serial}$  is defined in

Table 69.

- The parameter KeyValidityInfo shall indicate the validity period of the keying material as indicated by the parameters ValidNotBefore and ValidNotAfter, which indicate the beginning and the end of the validity period, respectively. The KeyValidityInfo shall be formatted as KeyValidityInfo =ValidNotBefore || ValidNotAfter, where:
- ValidNotBefore and ValidNotAfter shall be represented as specified in 12.22.4.3.
- The parameter Text shall be the representation of additional information, as specified in H.3.4.
- The string  $I_U$  as specified in the implicit certificate generation scheme (see H.5.1) shall be the octet string consisting of the octet strings Subject, Issuer, and Text, as follows:

$$I_U = \text{Subject} \parallel \text{Issuer} \parallel \text{Text}$$

### H.3.3 Elliptic curve cryptography manual certificates

Manual certificates  $MC_U$  shall be specified as  $MC_U = \text{PublicKey} \parallel \text{Subject} \parallel \text{Issuer} \parallel \text{Text}$ , where:

- The parameter PublicKey shall be the octet representation of the public key  $W_U$  as specified in the manual certificate generation transformation.
- The parameter Subject shall be the entity  $U$  of the purported owner of the private key corresponding to the public key represented by PublicKey.
- The parameter Issuer shall be the entity of the CA that creates the manual certificate during the execution of the manual certificate generation transformation (the so-called Certificate Authority).
- The parameter Usage\_Serial is defined in Table 69.
- The parameter KeyValidityInfo shall indicate the validity period of the keying material as indicated by the parameters ValidNotBefore and ValidNotAfter, which indicate the beginning and the end of the validity period, respectively. The KeyValidityInfo shall be formatted as KeyValidityInfo = ValidNotBefore || ValidNotAfter, where:
- ValidNotBefore and ValidNotAfter shall be represented as specified in 12.22.4.3.
- The parameter Text shall be the representation of additional information, as specified in H.3.4.
- The string  $I_U$  as specified in the manual certificate scheme (see H.10) shall be the octet string consisting of the octet strings Subject, Issuer, and Text, as follows:

$$I_U = \text{Subject} \parallel \text{Issuer} \parallel \text{Text}.$$

NOTE A manual certificate is not a real digital certificate, since the binding between the PublicKey and the Subject is established and verified by non-cryptographic means.

### H.3.4 Additional information

Additional information Text shall be specified as follows:

Text = Reserved, where:

- The parameter Reserved allows for future extensions of the additional information and shall be set to the all-zero bit string for this version of the standard.

## H.4 Key agreement schemes

### H.4.1 Symmetric-key key agreement scheme

The symmetric-key key agreement scheme used in this specification shall be the full symmetric-key with key confirmation scheme as specified with the following instantiations:

- Each entity shall be identified as specified in F.3.
- Each entity shall use the HMAC-scheme as specified in H.1.5.

- Each entity shall use the cryptographic hash function as specified in H.1.5.
- The parameter keydatalen shall have the same integer value as the key size parameter keylen as specified in H.1.2.
- Each entity shall use the challenge domain parameters as specified in H.1.8.
- All octets shall be represented as specified in F.2.

#### **H.4.2 Asymmetric-key key agreement scheme**

The asymmetric-key key agreement scheme used in this specification shall be the full MQV with key confirmation scheme as specified in 6.11 of ANSI X9.63:2001, with the following instantiations:

- Each entity shall be identified as specified in F.3.
- Each entity shall use the HMAC-scheme as specified in H.1.5.
- Each entity shall use the cryptographic hash function as specified in H.1.5.
- The parameter keydatalen shall have the same integer value as the key size parameter keylen as specified in H.1.2.
- The parameter SharedData shall be the empty string; parameter shareddatalen shall have the integer value 0.
- Each entity shall use the elliptic curve domain parameters as specified in H.2.2.
- All elliptic curve points shall be represented as specified in H.2.3.
- All octets shall be represented as specified in F.2.

### **H.5 Keying information schemes**

#### **H.5.1 Implicit certificate scheme**

The implicit certificate scheme used in this standard shall be the ECQV implicit certificate scheme (as specified in Standards for efficient cryptography, SEC 4: Elliptic curve Qu-Vanstone implicit certificate scheme, version 0.9, Certicom Research, November 14, 2007, available from <http://www.secg.org/>), with the following instantiations:

- Each entity shall be identified as specified in F.3.
- Each entity shall use the cryptographic hash function as specified in H.1.5.
- Each entity shall use the elliptic curve domain parameters as specified in H.2.2.
- All elliptic curve points shall be represented as specified in H.2.3.
- All implicit certificates shall be represented as specified in H.3.2.
- The implicit certificate infrastructure shall be one of the schemes as specified in H.3.2.
- All octets shall be represented as specified in F.2.

#### **H.5.2 Manual certificate scheme**

The manual certificate scheme used in this standard shall be the manual certificate scheme as specified in H.10, with the following instantiations:

- Each entity shall be identified as specified in F.3.
- Each entity shall use the elliptic curve domain parameters as specified in H.2.2.
- All elliptic curve points shall be represented as specified in H.2.3.
- All manual certificates shall be represented as specified in H.3.2.
- The manual certificate infrastructure shall be one of the schemes as specified in H.3.2.
- All octets shall be represented as specified in F.2.

## H.6 Challenge domain parameter generation and validation

### H.6.1 Overview

Challenge domain parameters impose constraints on the length(s) of bit challenges that a scheme expects. As such, this determine a bound on the entropy of challenges and, thereby, on the security of the cryptographic schemes in which these challenges are used. In most schemes, the challenge domain parameters will be such that only challenges of a fixed length will be accepted (e.g., 128-bit challenges). However, one may define the challenge domain parameters such that challenges of varying length might be accepted. The latter is useful in contexts wherein entities that wish to engage in cryptographic schemes might have a bad random number generator on-board. Allowing both entities that engage in a scheme to contribute sufficiently long inputs enables each of these to contribute sufficient entropy to the scheme at hand.

In this standard, challenge domain parameters will be shared by a number of entities using a scheme of the standard. The challenge domain parameters may be public; the security of the system does not rely on these parameters being secret.

### H.6.2 Challenge domain parameter generation

Challenge domain parameters shall be generated using the following routine:

- Input: This routine does not take any input.
- Actions: The following actions are taken:
  - Choose two nonnegative integers  $\text{minchallengelen}$  and  $\text{maxchallengelen}$ , such that  $\text{minchallengelen} \leq \text{maxchallengelen}$ .
- Output: Challenge domain parameters  $D=(\text{minchallengelen}, \text{maxchallengelen})$ .

### H.6.3 Challenge domain parameter verification

Challenge domain parameters shall be verified using the following routine:

- Input: Purported set of challenge domain parameters  $D=(\text{minchallengelen}, \text{maxchallengelen})$ .
- Actions: The following checks are made:
  - Check that  $\text{minchallengelen}$  and  $\text{maxchallengelen}$  are nonnegative integers.
  - Check that  $\text{minchallengelen} \leq \text{maxchallengelen}$ .
- Output: If any of the above verifications has failed, then output invalid and reject the challenge domain parameters. Otherwise, output valid and accept the challenge domain parameters.

## H.7 Challenge validation primitive

Challenge validation refers to the process of checking the length properties of a challenge. It is used to check whether a challenge to be used by a scheme in the standard has sufficient length (e.g., messages that are too short are discarded, due to insufficient entropy).

The challenge validation primitive is used in H.7 and uses the following routine:

- Input: The input of the validation transformation is a valid set of challenge domain parameters  $D=(\text{minchallengelen}, \text{maxchallengelen})$ , together with the bit string  $\text{Challenge}$ .
- Actions: The following actions are taken:
  - Compute the bit-length  $\text{challengelen}$  of the bit string  $\text{Challenge}$ .
  - Verify that  $\text{challengelen} \in [\text{minchallengelen}, \text{maxchallengelen}]$ . (That is, verify that the challenge has an appropriate length.)
- Output: If the above verification fails, then output invalid and reject the challenge. Otherwise, output valid and accept the challenge.

## H.8 Secret key generation (SKG) primitive

The SKG primitive derives a shared secret value from a challenge owned by an entity  $U_1$  and a challenge owned by an entity  $U_2$  when all the challenges share the same challenge domain parameters. If the two entities both correctly execute this primitive with corresponding challenges as inputs, the same shared secret value will be produced.

The shared secret value shall be calculated as follows:

- Prerequisites: The following are the prerequisites for the use of the SKG primitive:
  - Each entity shall be bound to a unique identifier (e.g., distinguished names). All identifiers shall be bit strings of the same length  $\text{entlen}$  bits. Entity  $U_1$ 's identifier will be denoted by the bit string  $U_1$ . Entity  $U_2$ 's identifier will be denoted by the bit string  $U_2$ .
  - A specialized<sup>12</sup> MAC scheme shall have been chosen, with tagging transformation as specified in 5.7.1 of ANSI X9.63:2001. The length in bits of the keys used by the specialized MAC scheme is denoted by  $\text{mackyelen}$ .
  - Input: The SKG primitive takes as input:
    - A bit string  $\text{MACKey}$  of length  $\text{mackyelen}$  bits to be used as the key of the established specialized MAC scheme.
    - A bit string  $\text{QEU}_1$  owned by  $U_1$ .
    - A bit string  $\text{QEU}_2$  owned by  $U_2$ .
  - Actions: The following actions are taken:
    - Form the bit string consisting of  $U_1$ 's identifier,  $U_2$ 's identifier, the bit string  $\text{QEU}_1$  corresponding to  $U_1$ 's challenge, and the bit string  $\text{QEU}_2$  corresponding to  $U_2$ 's challenge:
      - $\text{MacData} = U_1 \parallel U_2 \parallel \text{QEU}_1 \parallel \text{QEU}_2$ .
    - Calculate the tag  $\text{MacTag}$  for  $\text{MacData}$  under the key  $\text{MacKey}$  using the tagging transformation of the established specialized MAC scheme:
      - $\text{MacTag} = \text{MAC}_{\text{MacKey}}(\text{MacData})$ .
    - If the tagging transformation outputs invalid, output invalid and stop.
    - Set  $Z=\text{MacTag}$ .
    - Output: The bit string  $Z$  as the shared secret value.

## H.9 Block-cipher-based cryptographic hash function

The Matyas-Meyer-Oseas hash function is a cryptographic hash function based on block-ciphers. This hash function is defined for block-ciphers with a key size that is equal to the block size, such as AES-128, and with a particular choice for the fixed initialization vector  $/V$  (we take  $/V=0$ ). (For a more general definition of the Matyas-Meyer-Oseas hash function, See 9.4.1 of A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of applied cryptography, Boca Raton: CRC Press, 1997.)

The hash function is defined as follows:

- Prerequisites: The following are the prerequisites for the operation of Matyas-Meyer-Oseas hash function:
  - A block-cipher encryption function  $E$  shall have been chosen, with a key size that is equal to the block size. The Matyas-Meyer-Oseas hash function has a message digest size that

---

<sup>12</sup> This refers to a MAC scheme wherein the MAC function has the additional property that it is also pre-image- and collision-resistant for parties knowing the key (see also remark 9.8 of Menezes et al.). Such MAC functions allow key derivation in contexts where unilateral key control is undesirable.

is equal to the block size of the established encryption function. It operates on bit strings of length less than  $2^n$ , where n is the block size, in octets, of the established block-cipher.

- A fixed representation of integers as binary strings or octet strings shall have been chosen.
- Input: The input to the Matyas-Meyer-Oseas hash function is as follows:
- A bit string M of length l bits, where  $0 \leq l < 2^n$ .
- Actions: The hash value shall be derived as follows:
- Pad the message M according to the following method:
  - Right-concatenate to the message M the binary consisting of the bit 1 followed by k 0 bits, where k is the smallest non-negative solution to the equation

$$l+1+k \equiv 7n \pmod{8n}.$$

- Form the padded message M by right-concatenating to the resulting string the n-bit string that is equal to the binary representation of the integer l.
- Parse the padded message M as  $M_1 || M_2 || \dots || M_t$  where each message block  $M_i$  is an n-octet string.
- The output  $\text{Hash}_t$  is defined by
  - $\text{Hash}_0 = 0^{8n}$ ;  $\text{Hash}_j = E(\text{Hash}_{j-1}, M_j) \oplus M_j$  for  $j=1, \dots, t$ .

Here,  $E(K, x)$  is the ciphertext that results from encryption of the plaintext x, using the established block-cipher encryption function E with key K; the string  $0^{8n}$  is the n-octet all-zero bit string.

- Output: The bit string  $\text{Hash}_t$  as the hash value.

The cryptographic hash function operates on bit strength of length less than  $2^n$  bits, where n is the block size (or key size) of the established block cipher, in octets. For example, the Matyas-Meyer-Oseas hash function with AES-128 operates on bit strings of length less than  $2^{16}$  bits. It is assumed that all hash function calls are on bit strings of length less than  $2^n$  bits. Any scheme attempting to call the hash function on a bit string exceeding  $2^n$  bits shall output invalid and stop.

## **H.10 Elliptic curve cryptography manual certificate scheme**

### **H.10.1 Overview**

This subclause specifies the manual certificate scheme based on elliptic curve cryptography (ECC) used in this specification.

The manual certificate scheme is used by three entities – a Certificate Authority CA, a certificate requester U, and a certificate processor V, where U wishes to obtain a manual certificate from CA in order to convey U's associated public key to V.

The manual certificate scheme is described in terms of a certificate generation transformation and a certificate processing transformation. CA, U, and V use these schemes when they wish to communicate.

Prior to use of the scheme, U, V, and CA agree on the parameters with which the scheme shall be used. In particular, this includes U and V obtaining an authentic copy of CA's unique identifier.

CA executes the manual certificate generation transformation to compute an elliptic curve public-key pair for U and a manual certificate MC for this public key provided by CA. V executes the manual certificate processing transformation, to obtain U's purported static public key from U's purported manual certificate MC presented to V.

The manual certificate generation transformation yields a public-key pair and a certificate for this public key. This public-key pair shall be communicated to the purported holder in a secure

and authentic way. The mechanism by which this public-key pair is communicated is outside the scope of this standard.

The manual certificate processing transformation yields a static public key (and associated keying information) purportedly bound to the claimed holder; evidence that this public key is genuinely bound to this entity can, however, not be corroborated via processing of the manual certificate. Thus, with manual certificates, the binding of an entity and its public or private key cannot be verified, although one may obtain evidence that some entity that claims to be bound to the public key has indeed access to the corresponding private key, during cryptographic usage of the public key (e.g., via execution of a authenticated key agreement scheme or a signing transformation involving this public-key pair).

The manual certificate generation transformation is specified in H.10.2 and the manual certificate processing transformation is specified in H.10.3.

The prerequisites for the use of the scheme are:

- An infrastructure shall have been established for the operation of the scheme, including a certificate format, certificate processing rules, and unique identifiers. For an example of such an infrastructure, see IETF RFC 3280.
- Each entity has an authentic copy of the systems elliptic curve domain parameters  $D=(p,a,b,G,n,h)$  or  $D=(m,f(x),a,b,G,n,h)$ . These parameters shall have been generated using the parameter generation primitive in 3.1.1.1 or the primitive specified in 3.1.2.1, both of SEC1: Elliptic Curve Cryptography, version 1.0. Furthermore, the parameters shall have been validated using the parameter validation primitives in 3.1.1.2 or 3.1.2.2 of SEC1: Elliptic Curve Cryptography, version 1.0.
- Each entity shall be bound to a unique identifier (e.g., distinguished names). All identifiers shall be bit strings of the same length entlen bits. Entity U's identifier will be denoted by the bit string U. Entity V's identifier will be denoted by the bit string V. Entity CA's identifier will be denoted by the bit string CA.
- A cryptographic hash function Hash shall have been chosen for use with the ECQV implicit certificate generation scheme. Let hashlen denote the length in bits of the output value of this hash function.
- Each entity shall have decided how to represent elliptic curve points as octet strings (i.e., compressed form, uncompressed form, or hybrid form).
- A fixed representation of octets as binary strings shall have been chosen (e.g., most-significant-bit-first order or least-significant-bit-first order).

## **H.10.2 Elliptic curve cryptography manual certificate generation transformation**

### **H.10.2.1 General**

CA shall execute the following transformation to provide a manual certificate for U. CA shall obtain an authentic copy of U's identifier.

### **H.10.2.2 Inputs**

This routine does not take any inputs.

### **H.10.2.3 Ingredients**

The certificate generation transformation employs the key pair generation primitive in 3.2.1 of SEC1: Elliptic Curve Cryptography, version 1.0, and the manual certificate generation primitive of the established infrastructure.

### **H.10.2.4 Actions**

CA shall proceed as follows:

- Use the key pair generation primitive specified in 3.2.1 of SEC 1: Elliptic Curve Cryptography, version 1.0, to generate an ephemeral key pair  $(w_U, W_U)$  for the parameters D.

- Convert the elliptic curve point  $W_U$  to the octet string  $WEU$  as specified in 2.3.3 in SEC 1: Elliptic Curve Cryptography, version 1.0.
- Construct the octet string  $I_U$ , which is the to-be-conveyed-manual-certificate data.  $I_U$  shall contain identification information according to the procedures of the established infrastructure and may also contain other information, such as the intended use of the public key, the serial number of the manual certificate, and the validity period of the manual certificate. The exact form of  $I_U$  depends on the manual certificate format specified during the setup procedure.
- Construct the octet string  $MC_U$ , which is  $U$ 's manual certificate, according to the procedures of the established infrastructure.  $MC_U$  shall contain the octet strings  $I_U$  and  $WEU$  encoded in a reversible manner. The exact form of  $MC_U$  depends on the manual certificate format specified during the setup procedure.

#### **H.10.2.5 Output**

The output is  $MC_U$  as  $U$ 's manual certificate provided by CA.

### **H.10.3 Elliptic curve cryptography manual certificate processing transformation**

#### **H.10.3.1 General**

$V$  shall execute the following transformation to obtain  $U$ 's purported static public key from  $U$ 's purported manual certificate provided by CA.  $V$  shall obtain an authentic copy of  $U$ 's and CA's identifier.

#### **H.10.3.2 Input**

The input is  $U$ 's purported manual certificate  $MC_U$  provided by CA.

#### **H.10.3.3 Ingredients**

The manual certificate processing transformation employs the public key validation primitive in 3.2.2 of SEC 1: Elliptic Curve Cryptography, version 1.0, and the manual certificate validation primitive of the established infrastructure.

#### **H.10.3.4 Actions**

$V$  proceeds as follows:

- Verify the content of  $MC_U$  according to the established infrastructure. This includes verifying the contents of the certificate, such as the subject's name and the validity period. If the subject's name is unequal to  $U$ , output invalid and stop.
- Derive  $I_U$  from  $MC_U$ , according to the manual certificate format specified during the setup procedure.
- Derive CA's identifier from  $I_U$ , according to the certificate format specified during the setup procedure. If CA's identifier is unknown to  $V$ , output invalid and stop.
- Derive  $WEU$  from  $MC_U$ , according to the manual certificate format specified during the setup procedure.
- Convert the octet string  $WEU$  to the elliptic curve point  $W_U$  as specified in 2.3.4 of SEC 1: Elliptic Curve Cryptography, version 1.0
- Verify that  $W_U$  is a valid key for the parameters  $D$  as specified in 3.2.2 of SEC 1: Elliptic Curve Cryptography, version 1.0. If the validation primitive rejects the key, output invalid and stop.

#### **H.10.3.5 Output**

If any of the above verifications has failed, then output invalid and stop; otherwise, output valid and accept  $W_U$  as  $U$ 's purported static public key. ( $V$  may accept  $W_U$  as  $U$ 's genuine static public key provided  $U$  evidences knowledge to  $V$  of the corresponding private key  $w_U$  and provided  $V$  accepts  $U$  to be the only party that may have access to this private key.)

## Annex I (INFORMATIVE)

### Definition templates

#### I.1 Object type template

It is recommended that standard objects and their associated standard object identifiers be identified in a table for quick reference, as shown in Table 512. This indicates the information needed to define standard object types defined by this standard.

**Table 512 – Table of standard object types**

<b>Defining organization:</b>			
<b>Standard object type name (not expected to be transmitted, size not specified – check DD limits)</b>	<b>Standard object type identifier (non-negative)</b>	<b>Standard object identifier (non-negative), if applicable Used for mandatory objects with exactly one instance per device</b>	<b>Object description (not expected to be transmitted, size not specified – check DD limits)</b>

Elements of the table include:

- Standard object type name defines the name of the object.
- Standard object type identifier is the standard non-negative numeric identifier of the object type; uniquely identifies this object type.
- Standard object identifier, for standard object types that are required by a device and that may only be instantiated once, represents the standard non-negative numeric identifier for the object instance. This identifier is common to all devices. If 7 bits does not suffice, the high order bit of the first octet shall be set, and another octet shall be available to extend the value of the identifier.
- Object description is a description of the purpose and intent of this object.

#### I.2 Standard object attributes template

The template shown in Table 513 indicates the information needed to define the attributes of a standard object.

**Table 513 – Template for standard object attributes**

<b>Standard object type name:</b>				
<b>Standard object type identifier:</b>				
<b>Defining organization:</b>				
<b>Attribute name</b>	<b>Attribute identifier</b>	<b>Attribute description</b>	<b>Attribute data information</b>	<b>Description of behavior of attribute</b>
ObjectIdentifier	Key identifier	Unique identifier for the object	Type: Unsigned16. Classification: Constant Accessibility: N/A Initial default value: N/A Valid value set: 1-32,767	N/A
			Type: Classification: Accessibility: Initial default value: Valid value set:	
Reserved for future use	0			

Elements of the table include:

- Standard object type name defines the name of the object type.
- Standard object type identifier is the standard numeric identifier of the object type that uniquely identifies this object type. The value of this identifier fits into at most two octets.
- Defining organization is the organization defining this object (e.g., base standard, standard defined extension to the base standard object, industry specific profile (and which industry), special interest group (and which interest group)), or device vendor.
- Attrib name defines the name of the attribute.
- Attrib ID is the standard numeric identifier of the attribute. All attributes of an object are uniquely identified. If 7 bits does not suffice, the high order bit of the first octet shall be set, and another octet shall be available to extend the value of the identifier.
- Description is the description of the attribute.
- Type is the data type of the attribute. If the data may vary in size (such as for a variable size OctetString or a variable size VisibleString), then the maximum number of octets of data is indicated.
- Classification is the data classification (constant, static, static-volatile, dynamic, non-bufferable) of the attribute.
- Accessibility is how the attribute may be accessed remotely (e.g., Read only, or Read/write)
- Initial default value specifies the initial default value.
- Valid value set specifies the valid set of values for this attribute.

### I.3 Standard object methods

The template shown in Table 514 indicates the information needed to describe the methods of a standard object.

**Table 514 – Template for standard object methods**

<b>Standard object type name:</b>			
<b>Standard object type identifier:</b>			
<b>Defining organization:</b>			
<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>	
< name of method>			
<b>Input arguments</b>			
<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>	<b>Argument description</b>
...			
<b>Output arguments</b>			
<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>	<b>Argument description</b>
...			

Elements of the table include:

- Standard object type name defines the name of the object.
- Standard object type identifier is the standard numeric identifier of the object type that uniquely identifies this object type. The value of this identifier fits into at most two octets.
- Defining organization is the organization defining this object (e.g., base standard, standard defined extension to the base standard object industry specific profile (and which industry), special interest group (and which interest group)).
- Method name is the name of the method.
- Method ID is the numeric identification of the method. All methods of an object will have unique method identifiers. If 7 bits does not suffice, the high order bit of the first octet shall be set, and another octet shall be available to extend the value of the identifier.
- Method description is the description of the method.

- List of input parameters and their data types is a list of input parameters, their type and length (if not explicitly discernable from the type), a description of use, (how they are used when sent on a method invoke). These should be listed in order of transmission.

NOTE 1 For simplicity, all parameters are required. If there are situations where parameters can vary, separate methods should be identified to accommodate each variance.

- List of output parameters and their data types is a list of output parameters, their type and length (if not explicitly discernable from the type), a description of use (how they are used when sent on a method invoke). These should be listed in order of transmission.

NOTE 2 For simplicity, all parameters are required. If there are situations where parameters can vary, separate methods should be identified to accommodate each variance.

- Description of behavior describes the behavior of the object when this method is invoked.

#### I.4 Standard object alert reporting template

The template shown in Table 515 indicates the information needed to describe the alert reporting behavior of a standard object.

**Table 515 – Template for standard object alert reporting**

<b>Standard object type name(s):</b>					
<b>Standard object type identifier:</b>					
<b>Defining organization:</b>					
<b>Description of the alert:</b>					
Alert class (Enumerated: alarm or event)	Alert category (Enumerated: device diagnostic, comm. diagnostic, security, or process)	Alert type (Enumerated: based on alert category)	Alert priority	Value data type	Description of value included with alert
				Type:	
				Initial default value:	
				Valid value set:	

Elements of the table include:

- Standard object type name defines the name of the object.
  - Standard object type identifier is the standard numeric identifier of the object type that uniquely identifies this object type(s) that may report this alert. The value of this identifier fits into at most two octets.
  - Defining organization is the organization defining this object (e.g., base standard, industry specific profile (and which industry), special interest group (and which interest group)).
  - Description of the alert describes the semantic meaning of the alert.
  - Alert class indicates if this is an event (stateless) or alarm (state-oriented) type of alert.
  - Alert category indicates if this is a device related (e.g., a device specific diagnostic), communication related, security related, or process related alert.
- NOTE Only one category applies. Selection of the best fit for an alert may need to be discussed in order to be best established.
- Alert type is dependent on the alert category. See the alert reporting model in 12.8 for further details.
  - Alert priority is the priority of the alert.
  - Value and size is the size and value included in the alert report.
  - Description of value included in alert report is the description of the value, if a value is included in the alert report (e.g., for a process alarm that is a high alarm, this may be the process variable (PV)).
  - Accessibility defines if the attribute is readable, writeable, or both.
  - Initial default value indicates the initial default value of the attribute.

- Description of value set describes the set of values that may be taken on by this attribute.
- Description of behavior describes the behavior of this attribute (e.g., when particular value is written, or error conditions). Restrictions on use (e.g., operators should not write to this attribute) may be noted here.

## I.5 Data structure definition

The template for describing data structures that are used to define special data types is given in Table 516.

**Table 516 – Template for data structures**

Standard data type name:		
Element name	Element identifier	Element scalar type
		Type:
		Size:
		Classification:
		Accessibility:
		Default value :
		Valid value set:

## Annex J (INFORMATIVE)

### Operations on attributes

#### **J.1 Operations on attributes**

##### **J.1.1 General**

Attribute classification and accessibility dictate the operations permitted on a given attribute. Attribute classification and accessibility are described in 12.6.

##### **J.1.2 Attribute classification**

For a discussion of attribute classification, see 12.6.3.

##### **J.1.3 Retrieving, setting, and resetting attributes**

###### **J.1.3.1 General**

Attributes defined in the management objects can be accessed using the standard ASL-provided read or write services. Such operations enable configuration of the layers, as well as monitoring of their status. They can be used to retrieve, set / modify, and reset the values of attributes. The service primitives for these services, as well as the enumerated service feedback codes, are given in Clause 12.

Attributes can be reset using the write service by writing the default value to the relevant attribute. If a reset attribute is defined for a management object, it can be used to reset all the attributes in that management object that belong to certain classes of attributes.

More complex methods may be defined if necessary, but only if the equivalent results cannot be achieved using the more direct read / write services. A complex method may be warranted, for example, to replace a sequence of communication transactions in order to save energy. A complex method may also be warranted when synchronization issues may result if individual actions are used, rather than an atomic transaction set.

###### **J.1.3.2 Scheduled operations to enable synchronized cutover**

The generic method template Scheduled\_Write provided in Table 517 can be used to define a method for writing a value to an attribute at a scheduled TAI time. It can also be used to reset an attribute to its default value at a scheduled TAI time.

**Table 517 – Scheduled\_Write method template**

Method name	Method ID	Method description		
Scheduled_Write	<given in management object definition>	Method to write a value to an indicated attribute at an indicate TAI time		
<b>Input Arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
1	Attribute_ID	Data type: Unsigned16 <given in management object definition>	The attribute ID in the management object to which this method is being applied	
2	Scheduled_TAI_Time	Data type: TAITimeRounded	TAI time at which the value should be written to the attribute	
3	Value	Data type: <given in management object definition>	The value that needs to be written to the attribute	
<b>Output Arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
None				

The service feedback codes given in 12.17.4.2.2 are expected to be used to indicate if the method execution was successful or not. If not successful, this code provides information indicating why it was not successful.

#### J.1.4 Retrieving and setting structured attributes

The generic method templates Read\_Row and Write\_Row given in Table 518 and Table 519 can be used for defining methods that retrieve and set / modify the values of structured attributes. When the structured attribute is visualized as an information table, these methods allow access to a particular row based on one or more unique index field values. It is assumed that each table has at least one unique index field. The index field may either be a single element or the concatenation of a few elements in the row.

The input argument Scheduled\_TAI\_Time in the Write\_Row method template allows scheduled operation for a particular row of the structured attribute. A value of 0 for this argument indicates an immediate write operation.

**Table 518 – Read\_Row method template**

Method name	Method ID	Method description		
Read_Row	<given in management object definition>	Method to read the value of a single row of a structured attribute whose data is visualized as an information table		
<b>Input Arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
1	Attribute_ID	Data type: Unsigned16 given in management object definition>	The attribute ID in the management object to which this method is being applied	
2	Index_1	Data type of first index field of the structured attribute <given in management object definition>	The first index field in the structured attribute to access a particular row	
n+1	Index_n	Data type of nth index field of the structured attribute <given in management object definition>	The nth index field in the structured attribute to access a particular row	
<b>Output Arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
1	Data_Value	Data type: <given in management object definition>	An octet string that contains the data value	

**Table 519 – Write\_Row method template**

Method name	Method ID	Method description		
Write_Row	<given in management object definition>	Method to set / modify the value of a single row of a structured attribute whose data is visualized as an information table		
<b>Input Arguments</b>				
Argument #	Argument name	Argument type (data type and length)	Argument description	
1	Attribute_ID	Data type: Unsigned16 given in management object definition>	The attribute ID in the management object to which this method is being applied	
2	Scheduled_TAI_Time	Data type: TAITimeRounded	TAI time at which the value should be written to the row of the structured attribute	
3	Index_1	Data type of first index field of the structured attribute <given in management object definition>	The first index field in the structured attribute to access a particular row	
N+2	Index_n	Data type of nth index field of the structured attribute <given in management object definition>	The nth index field in the structured attribute to access a particular row	

	N+3	Data_Value	Data type: <given in management object definition>	An octet string that contains the data value
<b>Output Arguments</b>				
	Argument #	Argument name	Argument type (data type and length)	Argument description
	None			

The service feedback codes given in 12.17.4.2.2 are expected to be used to indicate if the method execution was successful or not. If not successful, this code provides information indicating why it was not successful.

A method based on the Write\_Row template can also be used to create a new row in the structured attribute if the index field(s) provided in the input argument(s) does/do not exist.

#### J.1.5 Resetting structured attribute values

For a structured attribute, the generic method template Reset\_Row given in Table 520 can be used for defining methods that reset / clear certain values in the structured attribute. The input argument Scheduled\_TAI\_Time in this method allows a scheduled reset operation. A value of 0 for this argument indicates an immediate reset operation.

**Table 520 – Reset\_Row method template**

Method name	Method ID	Method description		
Reset_Row	<given in management object definition>	Method to reset a single row of a structured attribute whose data is visualized as an information table		
<b>Input arguments</b>				
	Argument #	Argument name	Argument type (data type and length)	Argument description
	1	Attribute_ID	Data type: Unsigned16 given in management object definition>	The attribute ID in the management object to which this method is being applied
	2	Scheduled_TAI_Time	Data type: TAITimeRounded	TAI time at which the row of the structured attribute should be reset
	3	Index_1	Data type of first index field of the structured attribute <given in management object definition>	The first index field in the structured attribute to access a particular row
	n+2	Index_n	Data type of nth index field of the structured attribute <given in management object definition>	The nth index field in the structured attribute to access a particular row
<b>Output arguments</b>				
	Argument #	Argument name	Argument type (data type and length)	Argument description
	None			

The service feedback codes given in 12.17.4.2.2 are expected to be used to indicate if the method execution was successful or not. If not successful, this code provides information indicating why it was not successful.

#### J.1.6 Deleting structured attribute values

The generic method template Delete\_Row described in Table 521 can be used for defining methods that delete the values of structured attributes. The input argument Scheduled\_TAI\_Time in this method allows a scheduled delete operation. A value of 0 for this argument indicates an immediate delete operation.

**Table 521 – Delete\_Row method template**

<b>Method name</b>	<b>Method ID</b>	<b>Method description</b>		
Delete_Row	<given in management object definition>	Method to delete a single row of a structured attribute whose data is visualized as an information table		
	<b>Input arguments</b>			
	<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>	<b>Argument description</b>
	1	Attribute_ID	Data type: Unsigned16 given in management object definition>	The attribute ID in the management object to which this method is being applied
	2	Scheduled_TAI_Time	Data type: TAITimeRounded	TAI time at which the row of the structured attribute should be deleted
	3	Index_1	Data type of first index field of the structured attribute <given in management object definition>	The first index field in the structured attribute to access a particular row
	n+2	Index_n	Data type of nth index field of the structured attribute <given in management object definition>	The nth index field in the structured attribute to access a particular row
	<b>Output arguments</b>			
	<b>Argument #</b>	<b>Argument name</b>	<b>Argument type (data type and length)</b>	<b>Argument description</b>
	None			

The service feedback codes given in 12.17.4.2.2 are expected to be used to indicate if the method execution was successful or not. If not successful, this code provides information indicating why it was not successful.

## J.2 Synchronized cutover

A synchronized cutover capability is needed for some attributes and structured attributes that represent management information. For such an attribute, updates for the attribute value may be scheduled by indicating the TAI cutover time information; this operation may be accomplished by using one of the methods defined above. Such updates are sent to the management object for which this attribute is defined. The management object immediately validates whether the cutover is feasible, and, if feasible, arranges for the cutover to occur on schedule.

## Annex K (NORMATIVE)

### Standard object types

This annex specifies the standard object types defined by this standard. Each object type has three pieces of information to identify it:

- A corresponding standard object type identifier that identifies the standard defined base object type (example: analog input);
- A corresponding object standard subtype identifier that identifies the a standard subtype of a standard base type (example: analog input specialized for temperature); and
- A corresponding vendor subtype identifier that identifies a vendor specific subtype of either a standard base object or standard subtype.

Standard base objects shall have their object subtype identifier value equal to zero (0) and their vendor subtype identifier equal to zero (0).

A newer version of this standard that finds it necessary to extend the base object type definition of this standard may maintain the standard object identifier value and the subtype value of zero (0). This is permitted since the DMO contains an attribute to represent the version of the standard in use by the device, which can be thus be used to establish the base object type structure in use.

ISA100 industry profiles may define a standard object subtype as a standard object. Doing so creates a standard profile subtype. This standard provides a range of 1-255 to represent all such standard object subtype across all profiles.

Vendors may also subtype either a standard base object or a standard subtype object. This standard provides a range of 1-255 for vendor specific subtyping.

Object subtyping occurs when:

- One or more attributes is/are added to the base type;
- One or more methods is/are added to the base type;
- One or more alerts is/are added to the base type; or
- Any combination of the above.

Examples of object identification with subtyping follow:

- Analog input standard base object:
  - Object type identifier = 99
  - Object standard subtype identifier = 0
  - Vendor subtype identifier = 0
- Analog input temperature subtype object:
  - Object type identifier = 99
  - Object standard subtype identifier = (this standard defines (this standard's profile team), range 1-255)
  - Vendor subtype identifier = 0
- Vendor-specific analog input object:
  - Object standard type identifier = 99
  - Object standard subtype identifier = 0
  - Vendor-specific subtype identifier = (vendor defines, range 1-255)
- Vendor-specific analog input temperature object:

- Object standard type identifier = 99
- Object standard subtype identifier = n
- Vendor specific subtype identifier = (vendor defines, range 1-255)

Table 522 specifies standard object types.

**Table 522 – Standard object types**

<b>Object type</b>	<b>Defined by</b>	<b>Standard object type identifier (1 octet)</b>	<b>Standard object industry subtype identifier (1 octet)</b>	<b>Object identifier restrictions</b>
<b>Object types available to all UAPs</b>				
Null object	ISA100	0	0	Reserved
UAP management object	ISA100	1	0	This object is required for all UAPs, but is not required for the DMAP.
AlertReceiving object	ISA100	2	0	
UploadDownload object	ISA100	3	0	
Concentrator object	ISA100	4	0	
Dispersion object	ISA100	5	0	
Tunnel object	ISA100	6	0	
Interface object	ISA100	7	0	
Reserved for use by this standard for standard UAP objects	ISA100	8-50	0	Reserved for future standard object definitions for profile independent objects
Reserved for use by this standard	ISA100	51-95	0	Industry-specific types
<b>Process control industry object types</b>				
Analog input	ISA100	99	0	Analog input
Analog output	ISA100	98	0	Analog output
Binary input	ISA100	97	0	Binary input
Binary output	ISA100	96	0	Binary output
<b>DMAP object types</b>				
DMAP: Device management object (DMO)	ISA100	127	0	This object facilitates the management of the device's general device-wide functions
DMAP: Alert reporting management object (ARMO)	ISA100	126	0	This object facilitates the management of the device's alert reporting functions
DMAP: Device security management object (DSMO)	ISA100	125	0	This object facilitates the management of the device's security functions
DMAP: Data link layer management object (DLMO)	ISA100	124	0	This object facilitates the management of the device's data link layer
DMAP: Network layer management object (NLMO)	ISA100	123	0	This object facilitates the management of the device's network layer
DMAP: Transport layer management object (TLMO)	ISA100	122	0	This object facilitates the management of the device's transport layer
DMAP: Application sub-layer management object (ASLMO)	ISA100	121	0	This object facilitates the management of the device's application sub-layer
DMAP: Device provisioning object (DPO)	ISA100	120	0	This object facilitates the provisioning of the device before it joins the network
DMAP: Health reports concentrator object (HRCO)	ISA100	128	0	This object facilitates the periodic publication of device health reports to the system manager
Standard management objects	ISA100	119-114	0	
System time service object (STSO)	ISA100	100	0	This object facilitates the management of system-wide time information
Directory service object (DSO)	ISA100	101	0	This object facilitates the management of addresses for all existing devices in the network
System configuration object (SCO)	ISA100	102	0	This object facilitates the configuration of the system including contract establishment, modification and

<b>Object type</b>	<b>Defined by</b>	<b>Standard object type identifier (1 octet)</b>	<b>Standard object industry subtype identifier (1 octet)</b>	<b>Object identifier restrictions</b>
				termination
Device management service object (DMSO)	ISA100	103	0	This object facilitates device joining, device leaving, and device configuration
System monitoring object (SMO)	ISA100	104	0	This object facilitates the monitoring of system performance
Proxy security management object (PSMO)	ISA100	105	0	This object acts as a proxy for the security manager
Device provisioning service object (DPSO)	ISA100	106	0	This object facilitates device provisioning
Standard system management objects	ISA100	107-113	0	Reserved for standard management object type definitions. See Clause System management for details
<b>Vendor-defined types</b>				
Vendor-defined objects	Vendor	129-255	0	Reserved for use by implementers

Table 523 specifies standard object instances.

**Table 523 – Standard object instances**

<b>Object type</b>	<b>Defined by</b>	<b>Standard object type identifier (1 octet)</b>	<b>Standard object industry subtype identifier (1 octet)</b>	<b>Standard object identifier (1 octet)</b>	<b>Object identifier restrictions</b>
<b>Object types available to all UAPs</b>					
Null object	ISA100	0	0	0	Reserved
UAP management object	ISA100	1	0	1	This object is required for all UAPs, but is not required for the DMAP
UploadDownload object	ISA100	3	0	2	For UAP upgrade use only
<b>Process control industry object types</b>					
N/A					
<b>DMAP object types</b>					
DMAP: Device management object (DMO)	ISA100	127	0	1	This object facilitates the management of the device's general device-wide functions
DMAP: Alert reporting management object (ARMO)	ISA100	126	0	2	This object facilitates the management of the device's alert reporting functions
DMAP: Device security management object (DSMO)	ISA100	125	0	3	This object facilitates the management of the device's security functions
DMAP: Data link layer management object (DLMO)	ISA100	124	0	4	This object facilitates the management of the device's data link layer
DMAP: Network layer management object (NLMO)	ISA100	123	0	5	This object facilitates the management of the device's network layer
DMAP: Transport layer management object (TLMO)	ISA100	122	0	6	This object facilitates the management of the device's transport layer
DMAP: Application sub-layer management object (ASLMO)	ISA100	121	0	7	This object facilitates the management of the device's application sub-layer
DMAP: Upload/download object (UDO)	ISA100	3	0	8	This object facilitates the management of the device's upload/download functions
DMAP: Device provisioning object (DPO)	ISA100	120	0	9	This object facilitates the provisioning of the device before it joins the network
DMAP: Health reports concentrator object (HRCO)	ISA100	128	0	10	This object facilitates the periodic publication of device health reports to the system manager
<b>System management AP standard types</b>					
System time service object (STSO)	ISA100	100	0	1	This object facilitates the management of system-wide time information
Directory service object (DSO)	ISA100	101	0	2	This object facilitates the management of addresses for all existing devices in the network
System configuration object (SCO)	ISA100	102	0	3	This object facilitates the configuration of the system including contract establishment, modification and termination
Device management service object (DMSO)	ISA100	103	0	4	This object facilitates device joining, device leaving, and device configuration
System monitoring object (SMO)	ISA100	104	0	5	This object facilitates the monitoring of system performance
Proxy security management object (PSMO)	ISA100	105	0	6	This object acts as a proxy for the security manager
Upload/download object (UDO)	ISA100	3	0	7	This object facilitates downloading firmware/data to

<b>Object type</b>	<b>Defined by</b>	<b>Standard object type identifier (1 octet)</b>	<b>Standard object industry subtype identifier (1 octet)</b>	<b>Standard object identifier (1 octet)</b>	<b>Object identifier restrictions</b>
					devices and uploading data from devices
Alert receiving object (ARO)	ISA100	2	0	8	This object receives all the alerts destined for the system manager
Device provisioning service object (DPSO)	ISA100	106	0	9	This object facilitates device provisioning
Health reports concentrator object (HRCO)	ISA100	4	0	10	This object facilitates the periodic publication of device health reports to the system manager
<b>Vendor-defined types</b>					

## **Annex L (INFORMATIVE)**

### **Standard data types**

Table 524 specifies the standard data type identifiers for the standard data types. Standard data types are defined for constructs that are accessible using ASL services, such as read and write.

NOTE 1 Data structures may be defined for communication not accessible using ASL services. An example of such a data structure is one that is used as a parameter of a method, but not exposed as an ASL accessible object attribute.

NOTE 2 Type identifiers are leveraging type identifiers used in an existing IEC standard.

**Table 524 – Standard data types**

Data type	Defined by	Type identifier (Unsigned16)	Size (in octets)
<b>Reserved types</b>			
Invalid (type not specified)	ISA100	0	0
<b>AP standard data structure types</b>			
Communication association endpoint	ISA100	468	See Table 265
ObjectAttributeIndexAndSize	ISA100	469	See Table 264
Communication contract data	ISA100	470	See Table 266
Alert communication endpoint	ISA100	471	See Table 267
ObjectIDandType	ISA100	472	See Table 271
Unscheduled correspondent	ISA100	473	See Table 272
<b>Process control types</b>			
Process control value and status for analog value	ISA100	65	See Table 300
Process control value and status for binary value	ISA100	66	See Table 301
Process control scaling	ISA100	68	See Table 304
Process control mode	ISA100	69	See Table 302
<b>Alert descriptor types</b>			
Process control alarm report descriptor for analog with single reference condition	ISA100	498	See Table 270
Alert report descriptor (also used for process control binary alarms)	ISA100	499	See Table 269
<b>General communication / management types</b>			
Contract_Data	ISA100	401	See Table 30
Address_Translation_Row	ISA100	402	See Table 14
New_Device_Contract_Response	ISA100	405	See Table 31
Meta_Data_Attribute	ISA100	406	See Table 2
Security_Sym_Join_Request	ISA100	410	See Table 62
Security_Sym_Join_Response	ISA100	411	See Table 63
Security_Sym_Confirm	ISA100	412	See Table 67
Security_Pub_Join_Request	ISA100	415	See Table 71
Security_Pub_Join_Response	ISA100	416	See Table 71
Security_Pub_Confirm_Request	ISA100	417	See Table 73
Security_Pub_Confirm_Response	ISA100	418	See Table 73
Security_New_Session_Request	ISA100	420	See Table 82
Security_New_Session_Response	ISA100	421	See Table 83
Security_Key_and_Policies	ISA100	422	See Table 85
Security_Key_Update_Status	ISA100	423	See Table 86
DPSOWhiteListTbl	ISA100	440	See Table 425
NLContractTbl	ISA100	441	See Table 207
NLRouteTbl	ISA100	442	See Table 208
NLATTbl	ISA100	443	See Table 209

## Annex M (NORMATIVE)

### **Protocol identification values**

Table 525 lists the Uint8 protocol identification values. These are currently used in the tunnel object.

**Table 525 – Protocol identification values**

<b>Value</b>	<b>Protocol</b>
0	None
1	HART
2	FF H1
3	Modbus/RTU
4	PROFIBUS PA
5	CIP
6	WirelessHART
7-255	<reserved>

NOTE 1 These protocol identification values have been isolated into this annex in order to facilitate ease of maintenance.

NOTE 2 Value 0 for None should be preserved or tunnel functionality will be impaired.

## Annex N (INFORMATIVE)

### **Tunneling and native object mapping**

#### **N.1 Overview**

Tunneling involves the exchange of PDUs of one protocol by utilizing a second protocol. Most often these PDUs are application PDUs, but lower layer PDUs may also be exchanged. The PDU is encapsulated in the second protocol at an origination node and sent through the network to a termination node. With tunneling, what goes in one end comes out the other end, no more, no less.

Foreign protocol application communication (FPAC) is a more sophisticated PDU exchange mechanism. It involves the usage of additional mechanisms, including caching, compression, address translation, and proxy. As far as the application is concerned, the same PDUs are still exchanged between the origination node and the termination node as with tunneling. The difference is that the additional mechanisms act to improve energy efficiency and host system responsiveness.

#### **N.2 Tunneling**

Tunneling carries messages verbatim between endpoints of a tunnel. This standard provides tunneling that uses un-buffered client/server exchange of foreign PDUs between exactly two pre-configured tunnel endpoints. No interpretation of the PDU content is required. For most legacy protocols, this method will not be energy efficient, and some protocols may not operate properly due to variable or lengthy response times associated with sleeping devices. Regardless of the shortcomings, in many cases this will be the most expedient method for adapting existing devices and systems to this standard.

An extension of tunneling interprets the addressing within foreign PDUs to allow dynamic foreign PDU exchange with multiple endpoints.

#### **N.3 Foreign protocol application communication**

Tunneling is not an appropriate mechanism for most low-power wireless link applications. It is usually necessary to minimize PPDU overhead and the number of transactions in order to conserve energy stored in batteries or to operate within the power budget of scavenging and harvesting techniques. In addition, foreign protocols often have a need for fast response in order to avoid built-in timeouts. Devices in low-power wireless operation are most often in a sleep mode and thus cannot respond immediately.

FPAC increases energy efficiency and addresses potential timing issues by utilizing change-of-state transfer and caching to eliminate redundant transfer. Improvements in energy efficiency and performance are achieved by caching the information in the gateway, transferring information to the gateway only when it changes, and providing a heartbeat mechanism for integrity. This minimizes transfers initiated by the end devices (i.e., periodic publications), as well as minimizing transfers initiated by the foreign communication link (i.e., multi-master access through the gateway). In addition, this method can address foreign protocol timing requirements. Compared to tunneling, additional effort is necessary to translate the foreign protocol.

This standard provides support for FPAC that minimizes PPDU overhead using a combination of techniques:

- Encapsulation is limited to a single encapsulation. Protocol translators provide additional encapsulation across foreign links as necessary.
- Encapsulation is achieved through configuration agreement by carrying the foreign protocol within the protocol defined by this standard, rather than by carrying additional protocol headers. Mapping occurs as follows:
  - Transport supported relationships (publish/subscribe and client/server).

- Foreign addresses and native addresses.
- Length fields and integrity fields.
- This standard provides a native application service format for message exchange. Foreign protocols have their own service formats and message exchange protocols. The tunnel object allows the transfer of foreign APDUs with no extraneous overhead imposed by the native application service format.

This standard provides support for FPAC that minimizes transaction overhead using the following techniques:

- Distributed buffer caching mechanisms to minimize redundant transfer of unchanged data between gateways and end devices.
- Periodic, change-of-state (CoSt), and aperiodic transfer mechanisms.
- Watchdog timers to monitor endpoint and communication channel availability and assure data quality.

This standard provides support for FPAC that improves foreign protocol device access timing performance (and minimizes unnecessary transactions) by the provision of buffered device information through a gateway high side interface

NOTE Change of state (CoSt) should not be confused with class of service (CoS) defined within IEEE 802.1Q.

#### **N.4 Native object mapping**

This standard supports a native object format and messaging services. Automation-specific objects can be used to support protocol translation by utilizing these objects to perform a mapping of the foreign protocol into these objects and their messaging. Compared to the tunneling and FPAC methods, additional effort is necessary to translate the foreign protocol.

#### **N.5 Tunneling and native object mapping tradeoffs**

Native object mapping has a unique advantage in the ability to build a single standard-compliant end device for use with multiple foreign protocols. This is especially attractive for new devices.

Tunneling and FPAC have an advantage in simplicity for adapting wired automation devices through an adapter. Little, if any, translation may be required on either end.

Utilizing tunneling in conjunction with native object mapping is also useful. This allows common legacy functions to use native object mapping, while rarely used functions can be tunneled. This can lead to less total effort in protocol translation.

## Annex O (INFORMATIVE)

### Generic protocol translation

#### O.1 Overview

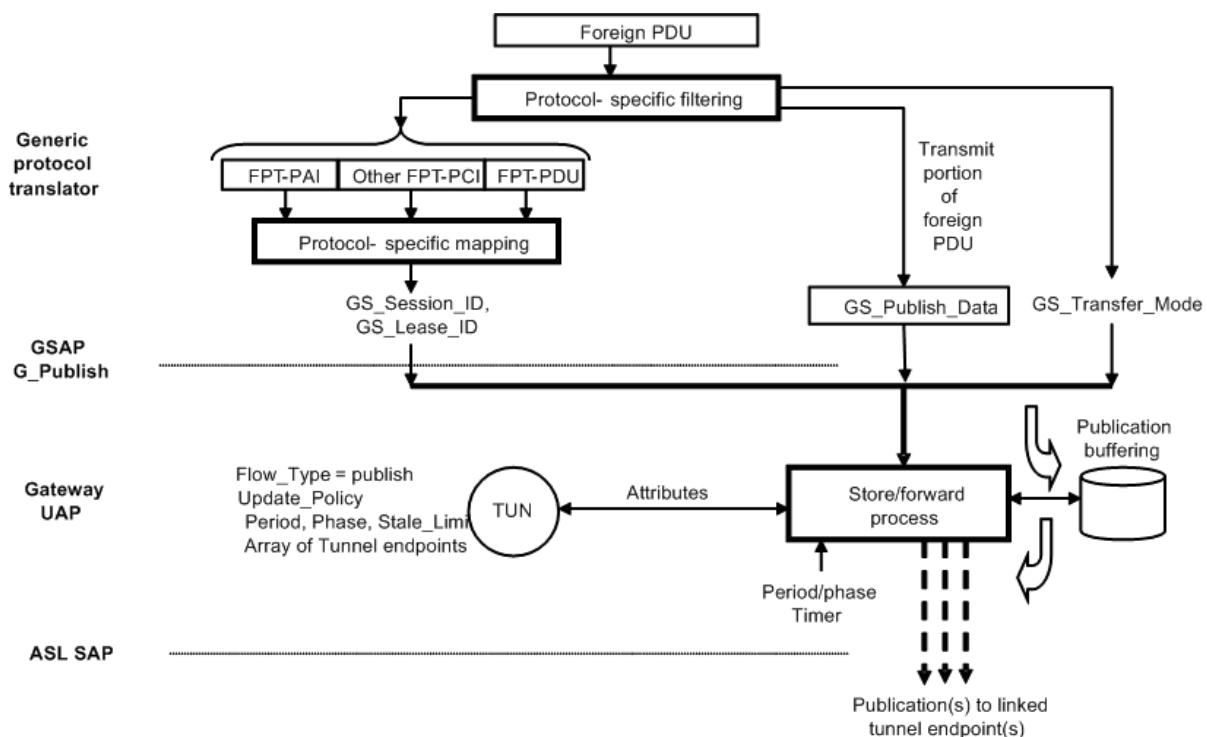
This standard does not include protocol translators. It does include supporting features that allow protocol translators to be constructed for common fieldbus protocols in a way that is compatible with low power wireless. The protocol translators are not defined herein and will be defined separately. As such, the protocol translation support is necessarily generic.

This annex is an example of how to use the tunnel object and the GSAP to support common protocol translation interactions. The tunnel object includes the normative features to support protocol translation.

Specific protocol translators (for specific fieldbuses) could include this annex, thus making it normative. They could also determine a different approach that still utilized the tunnel object and was compliant.

#### O.2 Publish

A portion of a generic gateway is depicted in Figure 170, which relates to the usage of publication. A generic protocol translator interacts with a gateway UAP through the GSAP. The gateway UAP utilizes the TUN object to interact with remote peers via the lower protocol suite through the ASL SAP.



**Figure 170 – Generic protocol translation publish diagram**

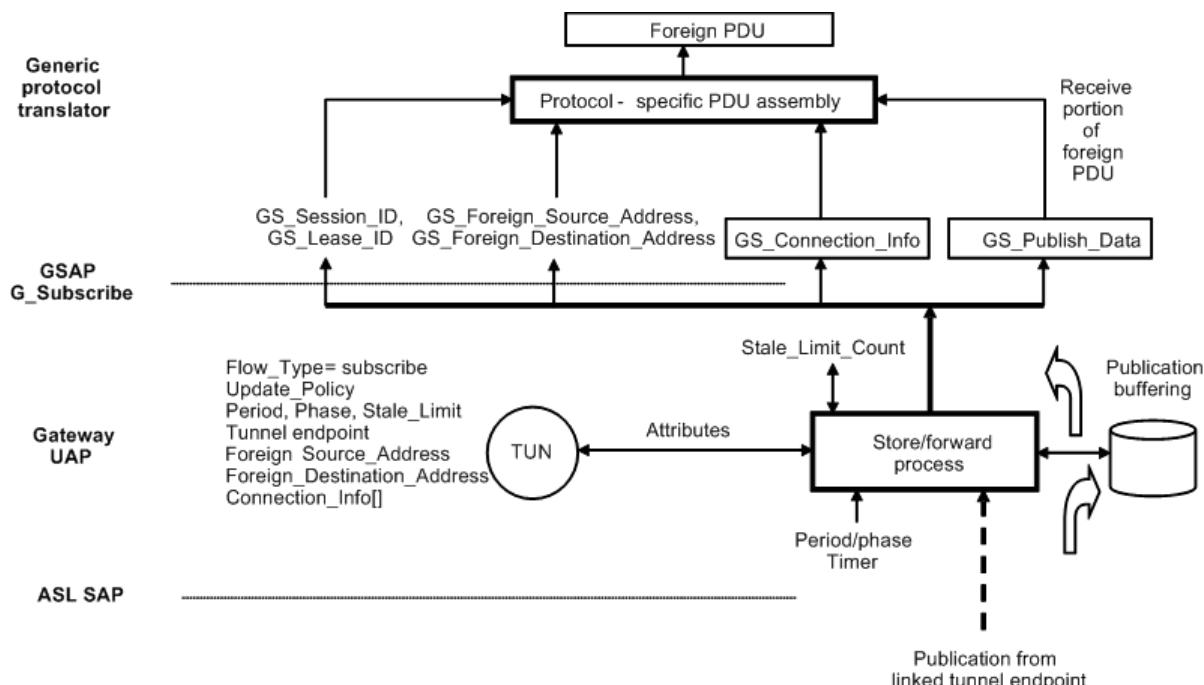
A foreign PDU is received by the protocol translator, and protocol-specific filtering is applied. Depending on the protocol, a combination of FPT-PAI, other FPT-PCI, and FPT-PDU may be necessary in order to determine the proper GS\_Session\_ID and GS\_Lease\_ID for GSAP usage in linking to a subscriber. The protocol-specific filtering determines the portion of the foreign PDU that needs to be transmitted (GS\_Publish\_Data) and foreign protocol-specific transport parameters such as priority (GS\_Transfer\_Mode). The parameters are then used to invoke GSAP services.

The GS\_Session\_ID and GS\_Lease\_ID are utilized by the gateway UAP to identify the TUN object and to retrieve the necessary parameters for store and forward processing decisions. GS\_Publish\_Data is buffered and forwarded at the appropriate time based on the Update\_Policy, the period, the phase, the Stale\_Limit, and the prior and current data content. Store and forward decisions are also driven by timer events based on the period and the phase. The ASL SAP is utilized to forward any messages.

A publication may be sent to one or more endpoints depending on the number of elements contained by the array of tunnel endpoints.

### O.3 Subscribe

A portion of a generic gateway is depicted in Figure 171, which relates to the usage of subscription. A generic protocol translator interacts with a gateway UAP through the GSAP. The gateway UAP utilizes the TUN object to interact with remote peers via the lower protocol suite through the ASL SAP.



**Figure 171 – Generic protocol translation subscribe diagram**

A publication APDU arrives at the gateway UAP through the ASL SAP. The addressing indicates a local TUN object that is linked to the remote publisher TUN object. The necessary attributes are retrieved from the TUN object for store and forward decisions.

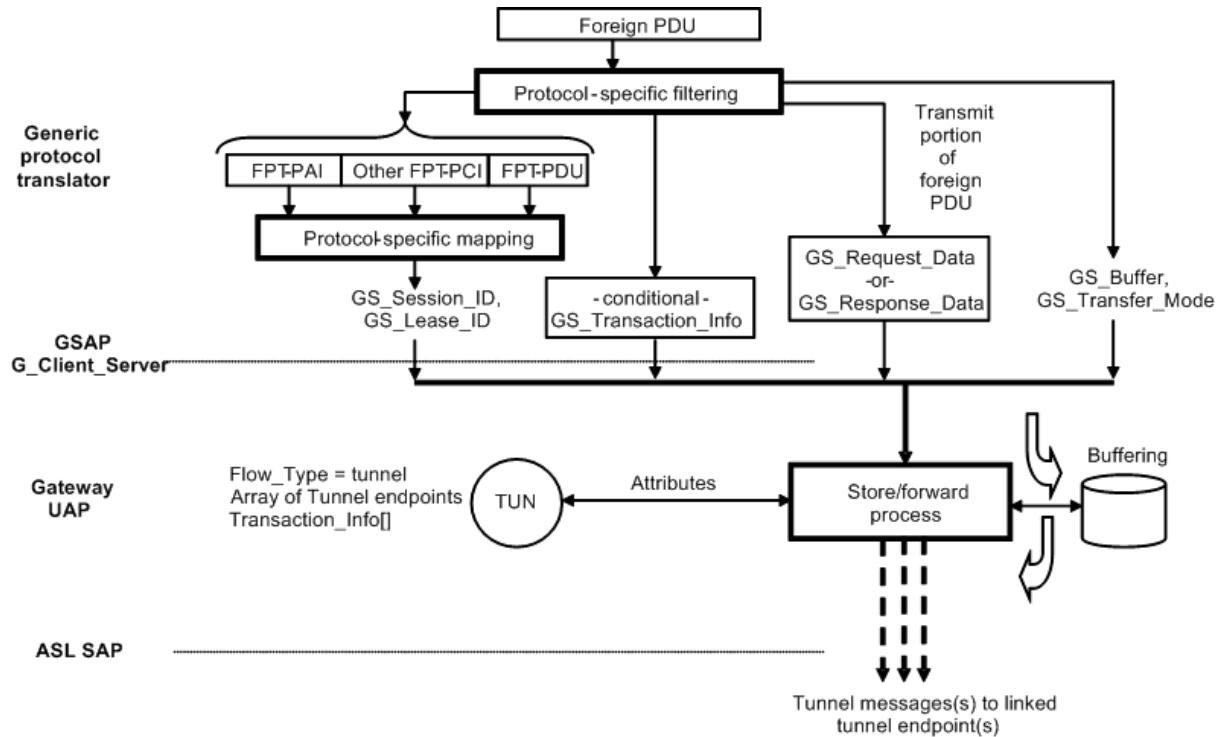
Publication data includes the GS\_Publish\_Data from the publisher. Publication buffering is based on Update\_Policy, the Period, the Phase, the Stale\_Limit, and Period/Phase based timer events. Forwarding occurs to the protocol translator through the GSAP based on polled and event driven interaction with the protocol translator. The gateway UAP also stores and includes the GS\_Session\_ID and GS\_Lease\_ID for the protocol translator to identify the publication. Publication specific information is optionally stored and provided to prevent unnecessary transmission of the information. This information includes addressing information (GS\_Foreign\_Source\_Address and GS\_Foreign\_Destination\_Address) and connection specific information (GS\_Connection\_Info).

The protocol translator performs a protocol-specific assembly to generate the foreign PDU.

### O.4 Client

A portion of a generic gateway is depicted in Figure 172, which relates to the transmission of client/server tunneled messages. A generic protocol translator interacts with a gateway UAP

through the GSAP. The gateway UAP utilizes the TUN object to interact with remote peers via the lower protocol suite through the ASL SAP.



**Figure 172 – Generic protocol translation client/server transmission diagram**

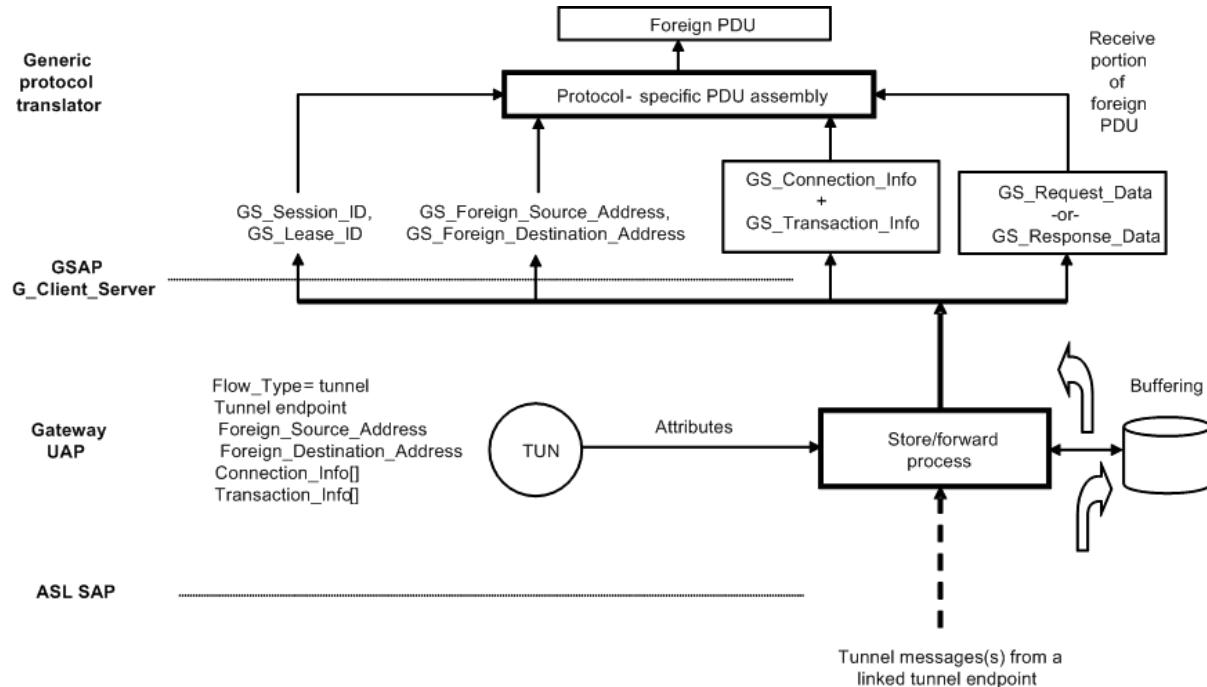
A foreign PDU is received by the protocol translator, and protocol-specific filtering is applied. Depending on the protocol, a combination of FPT-PAI, other FPT-PCI, and FPT-PDU may be necessary in order to determine the proper GS\_Session\_ID and GS\_Lease\_ID for GSAP usage. Protocol-specific filtering determines the portion of the foreign PDU that needs to be transmitted (GS\_Request\_Data or GS\_Response\_Data) and the appropriate transport parameters such as priority (GS\_Transfer\_Mode). For requests, an optional portion of the foreign PDU (GS\_Transaction\_Info) may also be provided for later return delivery through the GSAP when a matching response arrives. The parameters are then used to invoke GSAP services.

The GS\_Session\_ID and GS\_Lease\_ID are utilized by the gateway UAP to identify the TUN object and to retrieve the necessary parameters for store and forward processing decisions. The GSAP information (GS\_Request\_Data or GS\_Response\_Data) is optionally buffered and forwarded, depending on whether buffering is requested (GS\_Buffer), depending on the prior buffer content, and depending on whether a request or response is specified. The ASL SAP is utilized to forward any messages.

A tunnel request message may be sent to one or more endpoints depending on the number of elements contained by the array of tunnel endpoints. A tunnel response message can be sent to a single endpoint, but multiple responses can be sent to the same endpoint over time.

## O.5 Server

A portion of a generic gateway is depicted in Figure 173, which relates to the reception of client/server tunneled messages. A generic protocol translator interacts with a gateway UAP through the GSAP. The gateway UAP utilizes the TUN object to interact with remote peers via the lower protocol suite through the ASL SAP.



**Figure 173 – Generic protocol translation client/server reception diagram**

A tunnel request or response APDU arrives at the gateway UAP through the ASL SAP. The addressing indicates a local TUN object that is linked to a remote TUN object. The necessary attributes are retrieved from the TUN for store and forward decisions.

Tunnel APDU data includes either GS\_Request\_Data or GS\_Response\_Data. Depending on the tunnel mode, the response data may be buffered to answer subsequent requests from local buffers. Forwarding occurs to the protocol translator through the GSAP based on polled and event driven interaction with the protocol translator. The gateway UAP also stores and includes the GS\_Session\_ID and GS\_Lease\_ID for the protocol translator to identify the tunnel data. Tunnel message specific information is optionally stored and provided to prevent unnecessary transmission of the information. This information includes addressing information (GS\_Foreign\_Source\_Address and GS\_Foreign\_Destination\_Address), connection specific information (GS\_Connection\_Info) and transaction specific information (GS\_Transaction\_Info) for responses.

The protocol translator performs a protocol-specific assembly to generate the foreign PDU.

## Annex P (INFORMATIVE)

### **Gateway service access point adaptations for this standard**

#### **P.1 General**

This standard does not define functionality for a complete gateway. It does include supporting features that allow gateway construction by the addition of a protocol translator and a hardware interface and stack for a foreign network. This addition requires a separate effort to define the protocol translator.

This standard defines an informative gateway interface, the gateway service access point (GSAP). The GSAP is intended to be an abstraction of the underlying wireless system. In particular, it is intended to provide an abstraction for the wireless system described in this standard as well as the wireless system described in the WirelessHART specification.

This annex describes one way to implement the informative GSAP by utilizing this standard's normative objects and services. It is not a complete design, but a reference to aid understanding.

Specific gateways (for specific fieldbuses) could include this annex, thus making it normative. They could also determine a different approach that was compliant.

The GSAP services are implemented by a specialized UAP that utilizes native objects.

#### **P.2 Parameters**

GS\_Network\_Address is the 128-bit network address.

#### **P.3 Session**

The GSAP session service tracks resources and releases the resource when the session is closed or expires. Resources include communication contracts, bulk transfers in progress, buffered information, publication/subscribe/client/server resources in objects, and alert subscriptions.

#### **P.4 Lease**

The GSAP lease service allows allocation of resources and individual release when the lease is closed or expires. Resources include communication contracts and object resources for: bulk transfer, publish/subscribe, client/server, and alerts.

A lease differs from a communication contract in that a lease allocates resources both local to a gateway entity and optionally for resources corresponding to a related communication contract.

The specification of multiple network addresses within the GS\_Network\_Address\_List represents a multicast group. Specifying multiple addresses will result in a simulated multicast via multiple unicast operations. Even though this is a single lease, simulated multicast requires the allocation of multiple point to point contracts and simultaneous management of this contract set within the gateway.

GS\_Resource identifies the bulk transfer item for a lease (Destination\_Port and OID). GS\_Resource is also utilized in the linkage of matching sets of TUN objects and matching CON and DIS objects. A matched publisher and a subscriber(s) specify related values in lease creation. These values, along with the GS\_Network\_Address\_List, allow the Array of Tunnel endpoint to be filled on linked TUN objects and CON and DIS objects to be allocated and linked.

Subscriber leases require specification of GS\_Update\_Policy, GS\_Period, GS\_Phase, and GS\_Stale\_Limit.

## P.5 Device list report

There is no specific adaptation information for this item.

## P.6 Topology report

There is no specific adaptation information for this item.

## P.7 Schedule report

There is no specific adaptation information for this item.

## P.8 Device health report

There is no specific adaptation information for this item.

## P.9 Neighbor health report

GS\_Signal\_Strength maps to ED and GS\_Signal\_Quality maps to LQI as defined in 9.1.15.2.

## P.10 Network health report

There is no specific adaptation information for this item.

## P.11 Time

There is no specific adaptation information for this item.

## P.12 Client/server

### P.12.1 General

The GSAP client/server service utilizes the TUN object or the IFO, depending on the lease establishment.

### P.12.2 Native access

Where the lease establishment specifies GS\_Protocol\_Type = 0, the native protocol shall be configured through an IFO. GS\_Network\_Address\_List shall be empty. GS\_Lease\_Parameters shall contain only GS\_Transfer\_Mode in order to set default transfer quality of service and priority.

The GS\_Transfer\_Mode shall contain a priority bit and a discard eligibility bit as defined in Clause 12 for the read, write and execute services.

The payloads (GS\_Request\_Data and GS\_Response\_Data) shall conform to the native APDU formats. This includes only the ASL service types read, write, and execute. The IFO objects in gateways shall transfer these formats by utilizing the read, write, and execute services. GS\_Transfer\_Mode shall be provided with each transfer in order to indicate the quality of service and the priority associated with the transfer.

GS\_Buffer shall be utilized to request buffered and unbuffered behavior as appropriate to the ASL service and attribute classifications.

GS\_Transaction\_Info shall be empty.

The native client/server service shall be utilized to address native objects in the gateway.

### P.12.3 Foreign access

Where the lease establishment specifies GS\_Protocol\_Type not equal to 0, a foreign protocol shall be configured through a TUN object. GS\_Network\_Address\_List shall be supplied to establish the remote TUN endpoints. GS\_Resource is utilized to determine whether 2-part or 4-part tunnel services applies and to match the TUN endpoints within devices. A lone client or server lease establishes a 2-part tunnel. A pair of client and server leases with the same

GS\_Resource establishes a 4-part tunnel. GS\_Lease\_Parameters shall supply GS\_Connection\_Info on Server services as appropriate for the foreign protocol and GS\_Transfer\_Mode in order to set default transfer quality of service and priority.

The payloads (GS\_Request\_Data and GS\_Response\_Data) shall conform to the foreign APDU formats. This includes specification of foreign service types and service-specific fields. The TUN objects in gateways shall transfer these payloads by utilizing the 2-part tunnel and the 4-part tunnel services. GS\_Cache shall be utilized to request buffered and unbuffered behavior as appropriate to the TUN object configuration and the foreign protocol requirements. GS\_Transfer\_Mode shall be provided with each transfer in order to indicate the quality of service and the priority associated with the transfer.

The GS\_Transfer\_Mode shall contain a priority bit and a discard eligibility bit as defined in Clause 12 for the tunnel service.

GS\_Transaction\_Info shall be supplied on client services and returned on server services as appropriate for the foreign protocol.

## P.13 Publish/subscribe

### P.13.1 General

The GSAP publish/subscribe service utilizes the TUN object or CON and DIS objects, depending on the lease establishment.

### P.13.2 Native access

Where the lease establishment specifies GS\_Protocol\_Type = 0, the native application protocol will be published through the CON object and subscribed through the DIS object. GS\_Network\_Address\_List shall be empty. GS\_Lease\_Parameters shall contain only GS\_Transfer\_Mode in order to set default transfer quality of service and priority.

GS\_Network\_Address\_List shall contain a single item that is used to establish the publish and subscribe endpoints. GS\_Network\_Address determines the remote device address. GS\_Resource is used to determine the DIS object within this device. A local CON object is selected to be linked with the remote DIS object. GS\_Lease\_Parameters shall supply GS\_Update\_Policy, GS\_Period, GS\_Phase, and GS\_Stale\_Limit to establish the periodic or changes of state behavior for the CON and DIS objects. GS\_Connection\_Info shall be empty.

The publication payload (GS\_Publish\_Data) shall be sent and received in NativeIndividualValue or NativeValueList format. The CON and DIS objects in gateways shall transfer these formats by utilizing the publish service. GS\_Transfer\_Mode shall be provided with each transfer in order to indicate the quality of service and the priority associated with the transfer.

The GS\_Transfer\_Mode shall contain a priority bit and a discard eligibility bit as defined in Clause 12 for the publish service.

### P.13.3 Foreign access

Where the lease establishment specifies GS\_Protocol\_Type not equal to 0, GS\_Protocol\_Type is used to specify the foreign application protocol that will be published through the TUN objects. GS\_Network\_Address\_List shall be supplied to establish the remote TUN endpoints. GS\_Resource is utilized to match the TUN endpoints within devices. GS\_Lease\_Parameters shall supply GS\_Update\_Policy, GS\_Period, GS\_Phase, and GS\_Stale\_Limit to establish the periodic or changes of state behavior for Publish and Subscribe services. GS\_Lease\_Parameters shall supply GS\_Connection\_Info on Subscribe services as appropriate for the foreign protocol and GS\_Transfer\_Mode in order to set default transfer quality of service and priority.

The publication payload (GS\_Publish\_Data) shall be sent and received in non-native format. The TUN objects in gateways shall transfer these formats by utilizing the publish service. GS\_Transfer\_Mode shall be provided with each transfer in order to indicate the quality of service and the priority associated with the transfer.

The GS\_Transfer\_Mode shall contain a priority bit and a discard eligibility bit as defined in Clause 12 for the tunnel service.

#### **P.14 Bulk transfer**

The GSAP bulk transfer service is implemented through the bulk transfer protocol and IFO and UDO objects.

Bulk transfer is utilized for upload/download in half-duplex mode. An IFO acts as a client. UDOs act as servers. The UDO object identifier represents the target resource for the operation. A series of application level block transfers are controlled by the end objects to provide in-order error-free delivery of complete blocks of a negotiated size. There is no reliance on reliable transfer in lower layers. A multi-phase transfer protocol (open, transfer and close) is employed. A series of separate requests and responses track the total transfer size. Timing attributes are defined for the UDO to assist the client in determining timeout and retry policies and to avoid congestion errors. An upload or download operation may be closed due to errors on either end.

Lease establishment for bulk transfers will establish the necessary communication resources via a communication contract prior to bulk transfer primitive use.

The G\_Bulk\_Open request primitive is used to initiate a bulk transfer. The target device for a bulk transfer is addressed by the GS\_Network\_Address. This is the 128-bit network address. The target item for a bulk transfer is identified by GS\_Resource, which contains the Transport\_Port and the OID pointing to a specific UDO.

#### **P.15 Alert**

The GSAP alert service is implemented through the alert (alarms and events) services.

Lease establishment for alerts will establish the necessary communication resources via a communication contract to enable alert receipt. GS\_Alert\_Source\_ID specifies Transport\_Port, OID, and alert type.

#### **P.16 Gateway configuration**

There is no specific adaptation information for this item.

#### **P.17 Device configuration**

There is no specific adaptation information for this item.

## Annex Q (INFORMATIVE)

### **Gateway service access point adaptations for WirelessHART®**

#### **Q.1 General**

##### **Q.1.1 Overview**

This standard does not define functionality for a complete gateway. It does include supporting features that allow gateway construction by the addition of a protocol translator and a hardware interface and stack for a foreign network. This addition requires a separate effort to define the protocol translator.

This standard defines an informative gateway interface, the gateway service access point (GSAP). The GSAP is intended to be an abstraction of the underlying wireless system. In particular, it is intended to provide an abstraction for the wireless system described in this standard as well as the wireless system described in the WirelessHART specification.

This annex describes one way to implement the informative GSAP by utilizing the WirelessHART command set. It is not a complete design, but a reference to aid understanding.

Specific gateways (for specific fieldbuses) could include this annex, thus making it normative. They could also determine a different approach that was compliant.

##### **Q.1.2 Reference**

This annex references IEC/PAS 62591, an IEC document submitted by HCF that mirrors the WirelessHART specification.

The PAS references HCF enumeration tables that are not included in the PAS, but reside in WirelessHART document spec183.

##### **Q.1.3 Addressing**

WirelessHART device addressing and identification information includes:

- Nickname: a 2-byte short identifier for a device
- Unique ID: an 8-byte globally unique identifier formed by HCF OUI = 0x001B1E + 5 byte HART Unique ID confirming to EUI-64 format
- Long Tag: a 32-byte human readable string

The GSAP interface utilizes 128-bit (16-byte) logical addresses. Most WirelessHART commands utilize nicknames. WirelessHART gateways are required to implement command 841 (Read Network Device Identity) using nickname that returns a unique ID and a long tag for a nickname. Command 832 (Read Network Device Identity) converts the unique ID to the nickname and long tag of a device.

It is recommended to map the unique ID into the low bytes of the longer GSAP address.

##### **Q.1.4 Stack Interface**

WirelessHART describes its highest interface as an interface to the network layer (NL). The NL interface description receives parameters that it uses to invoke a transport layer (TL). Regardless of the interface description, the over the air packet encapsulates the TL header within a NL payload.

The TL payload encapsulates one or more HART or WirelessHART commands, both requests and responses. This annex describes the mapping of the GSAP services to commands that are carried by the TL.

### **Q.1.5 Tunneling**

WirelessHART gateways are required to tunnel HART commands. This means that a gateway includes a foreign network (the host interface) connected to the gateway and the gateway will tunnel HART commands through the foreign network.

### **Q.1.6 Entities**

The virtual gateway, network manager, host interface (host applications) and network interface (network devices) are all WirelessHART entities that implement (issue and respond to) HART and WirelessHART commands. The network manager has exclusive communication to a security manager. All communication between the network manager and the network devices and all communication between the host applications and the network devices is routed through the virtual gateway, which acts as a command routing hub. The virtual gateway itself also implements certain commands. The virtual gateway communicates to the network devices through one or more network access points as well as interposing network devices that perform routing.

### **Q.1.7 Delayed Response**

HART incorporates a delayed response mechanism, where a first response indicates that the command was received but that the actual response is delayed due to extended processing requirements. The GSAP services require handling of delayed responses within the gateway. An error is returned if a command that expects an acknowledgement is not acknowledged.

## **Q.2 Parameters**

GS\_Network\_Address shall be a 128-bit logical address used to identify a specific WirelessHART device within a network.

GS\_Undevice\_ID shall be a 64-bit device unique identifier in EUI-64 format used to identify a unique WirelessHART device. All gateways share a unique ID of 0xF981000002.

GS\_Network\_ID shall indicate a WirelessHART network that is accessible through the gateway. WirelessHART defines a 16-bit ID. WirelessHART specifies a single gateway per network. A multi-mode gateway specifies multiple networks per gateway and utilizes the network ID to identify the specific network associated with a WirelessHART virtual gateway.

## **Q.3 Session**

Multiple sessions may be established through a gateway. Each session is used to communicate with a specific network as indicated by the GS\_Network\_ID that is provided when the session is invoked.

WirelessHART includes a different concept that is also called a session. This session refers to an end to end security session. This clause does not refer to the security session, but the GSAP session.

The session service releases WirelessHART virtual gateway resources when a session ends explicitly or by timer expiration by utilizing the following commands:

- Release all leases
- Release unused communication resources
- Release unused cache

## **Q.4 Lease**

A lease is used to allocate and release specific communication resources within the context of a session.

NOTE WirelessHART “services” are allocated communication path resources from a requesting device (including the gateway) to a destination. Services are requested from the network manager and identified by a service ID. Services may vary in bandwidth and latency guarantees based on service allocation requests. The network manager handles establishment and management of intermediate resources based on requests. Resources may include common (shared) routes.

A lease is established with command 799 (request service). This command is used to request from the network manager a connection to another device (a service) with specified bandwidth and latency.

The service is identified by a service ID (maps to GS\_Lease\_ID).

GS\_Lease\_Period is set by the protocol translator.

GS\_Lease\_Type is defined by the service request flags and the service application domain. Service request flags (0x01 = Source, 0x02 = Sink, 0x04 = Intermittent) are specified in HCF enumeration table 39. The service application domain (0 = Publish, 1 = Event, 2 = Maintenance, 3 = Block transfer) is specified in HCF enumeration table 40.

GS\_Protocol\_Type is defined in Annex M.

The nickname specifies the address of the gateway peer for the service (maps to GS\_Network\_Address\_List which includes a single GS\_Network\_Address). WirelessHART includes multicast mechanisms, but not for services. Device level peer-to-peer is possible within the protocol, but not recommended due to security concerns.

GS\_Resource is unused in this context and shall be set to 0.

The period/latency maps to GS\_Lease\_Parameters (GS\_Period, GS\_Phase, and GS\_Stale\_Limit).

Command 801 (delete service) is used to notify a device of the deletion of a specific service (based on the service ID) due to peer request or network manager decision.

## **Q.5 Device list report**

A WirelessHART gateway is required to implement command 814 (read device list entities). This command retrieves a list of the unique IDs for the devices known to the gateway.

All devices returned are on the active device list. Whitelist and blacklist indication are maintained in the network manager and within the gateway.

GS\_Network\_Address, GS\_Under\_Device\_ID, GS\_Manufacturer, GS\_Model, and GS\_Revision are returned for each device.

## **Q.6 Topology report**

The topology report shall return a list of devices (GS\_Device\_List), their address (GS\_Network\_Address), and related information. The device list report shall be implemented to identify the devices in a system.

A WirelessHART gateway is required to implement command 834 (read network topology information). This command is used to retrieve the graph information (GS\_Graph\_List) for a specific device. Retrieved information includes a list of Graph IDs (GS\_Graph\_ID) for the graphs that the device participates in and a list of nicknames for the neighbors in the graph (associated to GS\_Network\_Address).

A WirelessHART gateway is required to implement command 833 (read network device's neighbor health), which returns the set of neighbors of a specific device. Each element in the list returns the neighbor nickname (which maps to GS\_Network\_Address within GS\_Neighbor\_List).

## **Q.7 Schedule report**

The schedule report service shall return schedule information for a specific device identified by GS\_Network\_Address. The device list report may be used to identify the devices in the system.

Command 783 (read superframe list, normally used by the network manager) shall be used to retrieve the list of superframes and their related information from a specific device. Retrieved

information includes the superframe ID (GS\_Superframe\_ID), the number of slots (GS\_Num\_Time\_Slots) and superframe mode flags (HCF enumeration table 47).

GS\_Slot\_Length is fixed to 10ms. GS\_Start\_Time is calculated from SuperframeSlot = (Absolute Slot Number) % Superframe.NumSlots.

Command 784 (read link list; normally used by the network manager) is used to retrieve information about the link entries from a specific device. Link entries are related to slot usage within superframes. Retrieved information includes the Superframe ID (GS\_Superframe\_ID), the slot number in the superframe, the channel (GS\_Channel), linkOptions (HCF enumeration table 46 maps to GS\_Direction), linkType (HCF enumeration table 45 maps to GS\_Link\_Type), and nickname (associated to GS\_Network\_Address) of the link neighbor to build GS\_Link\_List.

GS\_Channel\_List contains a list of whitelist and blacklist channels as defined by GS\_Channel\_Status to reach GS\_Channel\_Number. GS\_Channel\_Number maps to Index = 0, IEEE Std 802.15.4 channel = 11, 2,405 MHz ... Index = 14, IEEE Std 802.15.4 channel = 25, 2,475 MHz. Command 817 (read channel blacklist) shall be used to identify the GS\_Channel\_Status for each channel.

## **Q.8 Device health report**

The device health report shall return device health information for a list of devices (GS\_Device\_List) each identified by GS\_Network\_Address.

All WirelessHART devices implement and periodically publish command 779 (report device neighbor health) to make information available to the network manager and applications.

A WirelessHART gateway is required to implement command 840 (read network device's statistics), which reports most of the command 779 information (no power status). This command uses a Unique ID to retrieve a variety of information related to a specific device, including:

- Number of packets generated by this device (GS\_DPDUs\_Transmitted)
- Number of packets terminated by this device (GS\_DPDUs\_Failed\_Transmission)
- Number of DL MIC failures (GS\_DPDUs\_Received, GS\_DPDUs\_Failed\_Reception)
- Number of NL MIC failures (GS\_DPDUs\_Received, GS\_DPDUs\_Failed\_Reception)
- Number of CRC errors (GS\_DPDUs\_Received, GS\_DPDUs\_Failed\_Reception)

Command 840 is used multiple times to gather information for each device in the list.

## **Q.9 Neighbor health report**

Neighbor health is periodically published to the network manager by command 780 (report neighbor health list). Neighbor signal strength is periodically published to the network manager by command 787 (report neighbor signal levels), which duplicates information in command 780.

G\_Neighbor\_Health\_Report returns a list of link-level connection quality information for the set of neighbors of a specific device. The service is primarily implemented by command 833.

A list of devices known to the gateway (and each device address GS\_Network\_Address) may be retrieved by using the GSAP device list report service (G\_Device\_List\_Report).

A WirelessHART gateway is required to implement command 833 (read network device's neighbor health) which returns a list of link-level connection quality information for the set of neighbors of a specific device. Each element in the list returns the neighbor nickname (which maps to GS\_Network\_Address), the receive signal level in dB (GS\_Signal\_Strength), the number of packets transmitted to the neighbor (GS\_DPDUs\_Transmitted), the number of failed transmissions to the neighbor where no ACK was received (GS\_DPDUs\_Failed\_Transmission), and the packets received from neighbor (GS\_DPDUs\_Received).

GS\_Link\_Status = 1 indicates that the neighbor is available for communication. GS\_Link\_Status = 0 indicates that the neighbor is unavailable for communication.

A WirelessHART gateway is required to implement command 840 (read network device's statistics), which reports GS\_DPDUs\_Failed\_Reception as described in the device health report clause.

GS\_Signal\_Quality is not available and shall be set to the maximum quality value.

## **Q.10 Network health report**

The device health report and neighbor health report are used to determine GS\_Device\_Health\_List and GS\_Network\_Health.

A WirelessHART gateway is required to implement command 840 (read network device's statistics). This command uses a Unique ID to retrieve a variety of information related to a specific device, including:

- Number of joins (GS\_Join\_Count)
- Date of most recent join and time of join (GS\_Start\_Date)
- Average latency from the gateway to this node (GS\_GPDU\_Latency)

ASN is a count of all slots that have occurred since forming the network. It always increments and is never reset. ASN is 5-bytes long. ASN 0 is when the network is born. GS\_Start\_Date and GS\_Current\_Date are derived from ASN.

## **Q.11 Time**

WirelessHART network time is measured relative to the absolute slot number 0 (ASN 0), which is the time when the network was born. Time advances in 10 ms increments per slot.

Time distribution is configured by the network manager by using command 971 (write neighbor property flag) to specify a neighbor with the neighbor flags (0x01 time source, HCF enumeration table 59) indicating a specific neighbor as a time source. The WirelessHART gateway is always configured as the source of network time.

Slot time is updated through neighbors by synchronization via time errors seen in packet exchanges (ACK TsError field).

The virtual gateway is required to synchronize with an external time source at least once per hour. UTC time is mapped to slot time from an external reference through the gateway. The mapping of ASN 0 to UTC is broadcast from the gateway. Command 793 (write UTC time mapping) is a gateway command that allows the network manager to set the mapping of the start of ASN 0 to UTC time on a device.

GS\_Time is based on TAI time. UTC time is based on TAI time with leap seconds added at irregular intervals. This service applies time updates through the GSAP. TAI and UTC time updates occur due to drift. UTC adds additional updates due to leap seconds. A conversion is necessary to the internal HART time format from and to GS\_Time: HART date 3 bytes, time of day, 3 bytes.

Command 794 (read UTC time mapping) is a gateway command that allows a device or the network manager to set and read the mapping of the start of ASN 0 to UTC time. GS\_Command is used to set and read GS\_Time within the gateway for synchronization purposes. Command 89 (Set Real-Time Clock) is used to set the time. Command 90 (read real-time clock) is used to read the current time.

## **Q.12 Client/server**

Unless specified elsewhere in this annex, the gateway shall tunnel all HART commands through the GSAP client/server service. These commands are issued from a master to a slave (field device). The master assumes the client role and the slave assumes the server role.

The commands follow a request/response format. Request data bytes are sent from the client to the server in GS\_Request\_Data. Response data bytes are returned from the server to the client in GS\_Response\_Data. The command-specific response codes are mapped into GS\_Status.

The GS\_Buffer flag is set or cleared to indicate whether a command is to be buffered. The following commands are buffered:

- 0 read unique id
- 11 read unique id associated with tag
- 13 read tag, descriptor, date
- 20 read long tag
- 21 read unique id associated with long tag?
- 48 read additional status
- 50 read dynamic variable assignments
- 18 write tag, descriptor, date
- 22 write long tag
- 25 write primary variable range values
- 44 write primary variable units

Multiple server responses may be received with the same GS\_Transaction\_ID in the case of a delayed response.

Client/Server priority is established via the GS\_Transfer\_Mode.

WirelessHART priority falls into one of four levels, command (highest priority), Process data, normal, and alarm (lowest priority). Command priority is reserved for packets containing network control, configuration and diagnostics. Process-data priority packets contain process data and are refused when three-quarters of a device's packet buffers are full. Alarm priority packets contain alarms and events. Only a single alarm priority packet is buffered. Normal priority packets are all other packets and are refused when one-half of a device's packet buffers are full.

GS\_Transaction\_Info is not required.

## **Q.13 Publish/subscribe**

The GSAP publish/subscribe service is implemented through publication of commands by the WirelessHART devices utilizing burst mode. Adapters are able to publish on behalf of non-native sub-devices. WirelessHART natively aggregates published commands where the time aligns and command 78 (read aggregated commands) is not required.

Normally, a gateway subscribes to a device publication. Within G\_Subscribe, GS\_Publish\_Data returns the published data.

It is required that a lease be acquired for the subscription (obtain GS\_Lease\_ID). Lease establishment allocates resources between the gateway and the device using command 799.

The G\_Publish\_Watchdog indication is received if the publication is not received by the GS\_Stale\_Limit.

### **Q.13.1 Lease Establishment**

A subscription lease is established through the lease service. GS\_Resource specifies the subscription information (command number and process variable list) to the lease service.

Command 108 (write publish data mode command number) is used to select the command to be published.

If command 108 specifies universal command 9 (read device variables) or common practice command 33 (read device variables), process variables will be assigned to slots for publication. Command 107 (write publish data device variables) is used to assign the slots.

Command 103 (write publish data period) selects the minimum (GS\_Period, GS\_Phase) and maximum update period (GS\_Stale\_Limit) for a publication (in 1/32 ms increments up to 3600 s; requested and actual values may differ).

Command 104 (write publish data trigger) sets a trigger condition (GS\_Update\_Policy) for publication (continuous/windowed/rising and a level) resulting in dynamic changes to publication time. Publication occurs at least as often as when the maximum period is reached.

Command 109 (publish data mode control) turns publishing on and off. The publication source device contacts the network manager to request bandwidth.

### **Q.13.2 Buffering**

The following commands are buffered:

- 1 read primary variable
- 2 read current & percent
- 3 read all variables
- 9 device variables and status
- 33 read device variables
- 123 read trend
- Device-specific

### **Q.14 Bulk transfer**

The GSAP bulk transfer service corresponds to the application layer provided block transfer. Operation permits upload/download (GS\_Mode) in either half or full duplex modes, and relies on the transport layer to provide a series of application level block transfers. The transport segments and reassembles based on limited MTU in lower layers and provides error free delivery of complete blocks (all pieces are in order).

Operation utilizes several phases, including open (G\_Bulk\_Open), transfer (G\_Bulk\_Transfer), reset, and close (G\_Bulk\_Close). New commands were created to execute these phases. A master opens a session (command 111) with a slave to initiate the operation (GS\_Transfer\_ID links the phases of this operation). The master proposes the block sizes (GS\_Block\_Size), and the slave may reduce the size. A port (a byte) identifies the target resource (GS\_Resource) for the operation (firmware, parameters, and log file). The total size is not stated (GS\_Item\_Size = 0) and may not be known even to the application (such as a continuous stream of samples organized in blocks). There is a byte counter selected by each end to track progress. Command 112 is organized such that the request contains download data (GS\_Bulk\_Data) and the response has upload data (GS\_Bulk\_Data). The request creates an indication in the slave; the response contains an indication in the master. The session is closed on errors. No rule exists on how to deal with the partial data set. The delayed response mechanism is mentioned in status, but is not described further.

### **Q.15 Alert**

The GSAP alert service is implemented through several mechanisms. Locally buffered changes include burst mode updates (process changes), event notification (general alarms and events), device status changes, device configuration changes, network topology changes, and network schedule changes.

Change notification simply indicates a change, and further action is required to retrieve altered information from the gateway buffers. The gateway entity acknowledges event arrival to devices. Publications and alerts are stored in the gateway entity. The gateway entity acknowledges alert arrival to devices.

For example, the gateway may internally use HART command 115-118 to set up change notification and HART command 119 to indicate that changes have occurred.

Events are configured with assigned event numbers on a per-device basis.

Command 116 (write event notification bit mask) configures the event mask that is used to trigger an event notification for a specific event. The event mask corresponds to command 48 (read additional device status), which refers to common tables 14, 17, 29, 30, 27, 31, 32, 28 and device specific status.

Command 117 controls the timing of event notifications. Event notification uses burst mode for delivery when an event is triggered. A de-bounce period is specified to prevent events that are too short from triggering a burst message. A retry time (desired burst period) and a maximum update time (maximum burst period) sets the burst transfer timing if an event triggers a message.

Command 118 (event notification control) is used to enable or disable an event notification for a specific event.

Command 119 (acknowledge event notification) is used to acknowledge the event notification and clear the event from being sent in the burst updates. Other events may be in queue.

Command 115 (read event notification summary) is used to determine the configuration of an event based on a specific event number.

The following commands are buffered:

- 119 read event notification status (time stamp + device status + command 48)
- Command 788 (alarm path down), command 789 (alarm source route failed), command 790 (alarm graph route failed), and command 791 (alarm transport layer failed) report communication failures to the network manager.

WirelessHART gateway command 836 (write update notification bit mask for a device) registers a client for notification updates. The device is addressed by the unique ID and given a set of change notification flags as defined in HCF enumeration table 60. Codes exist for BurstMode, EventNotification, DeviceStatus, DeviceConfiguration, NetworkTopology (gateway or NM), and NetworkSchedule (gateway or NM). This is used in G\_Alert\_Subscription to subscribe (by providing a GS\_Subscription\_List with GS\_Alert\_Source ID, GS\_Subscribe, and GS\_Enable for a specific device GS\_Network\_Address and a specific category GS\_Category).

WirelessHART gateway command 838 (read update notification bit mask for a device) returns a list of the update notifications for a device. This is used in G\_Alert\_Subscription to identify the subscriptions.

WirelessHART gateway command 839 (change notification) is sent by the gateway to a client and returns a list of up to 10 change notifications (cached commands) for a device. Each change results in a single G\_Alert\_Notification.

## **Q.16 Gateway configuration**

There is no specific adaptation information for this item.

## **Q.17 Device configuration**

There is no specific adaptation information for this item.

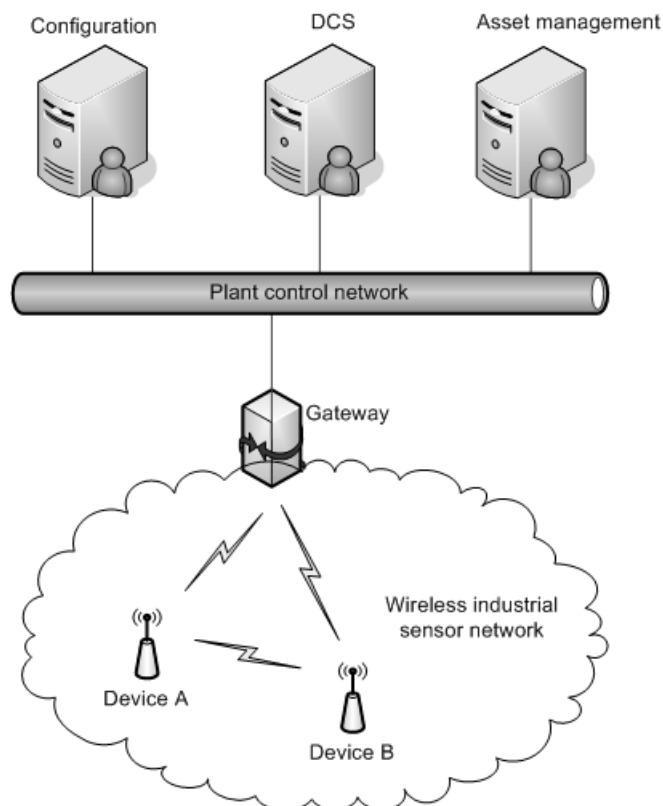
## Annex R (INFORMATIVE)

### **Host system interface to standard-compliant devices via a gateway**

#### **R.1 Background**

##### **R.1.1 Host system integration reference model**

A simplified reference model for standard-compliant device/host system integration is depicted in Figure 174.



**Figure 174 – Host integration reference model**

##### **R.1.2 Asset management tools**

Asset management involves overseeing the health of the systems assets by monitoring health related conditions in order to identify a potential problem before the process or plant operation is affected. Host systems provide an asset management tool or set of tools to fulfill the asset management function, with goals of lowering maintenance costs, reducing down-time, and ensuring that appropriate product quality levels are met.

##### **R.1.3 Configuration tools**

Once the system design has been established, and the system components identified, the operation of the components in overall system needs to be configured. Host systems provide a configuration tool or set of tools that support system component configuration and define component operation in the system.

##### **R.1.4 Distributed control system**

A distributed control system (DCS) is a control system that supports a process wherein the control elements are geographically distributed. These distributed elements are connected by communication networks, which are used for communicating with the distributed elements.

### R.1.5 Gateway

A gateway connects the host systems with the network. See Clause 13 for more information regarding the gateway.

## R.2 Device application data integration with host systems

### R.2.1 General

There are two generic integration options for integrating application data from connected devices with host systems:

- Integration via protocol mapping; and
- Integration via protocol tunneling.

### R.2.2 Native protocol integration via mapping

Existing host systems may integrate device application data by mapping the relationship between the devices and data to the information handling performed by the existing host system. This mapping function is typically performed by a gateway between the existing host system and the wireless industrial sensor network (WISN).

### R.2.3 Legacy device protocol integration via tunneling

Existing host systems may integrate application data from existing legacy devices that are using the WISN application tunneling capability in the same manner by which it presently integrates the application data from the legacy devices.

## R.3 Host system configuration tool

### R.3.1 General

Host systems typically support either one or both of two generic integration options for configuring field devices:

- Electronic device description language (EDDL)
- Field device tool / device type manager (FDT/DTM)

### R.3.2 Host configuration using electronic device description language

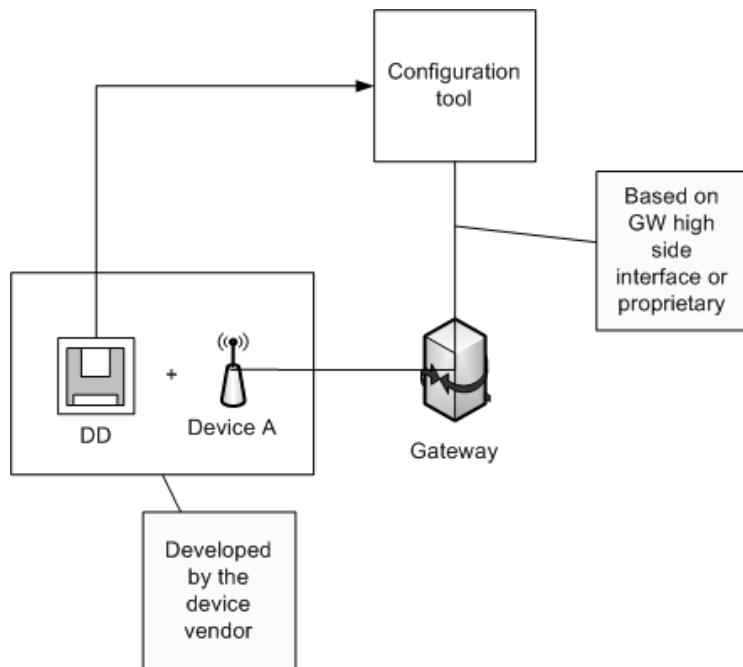
EDDL is an IEC standard, IEC 61804-3, that describes a generic language for describing automation device properties. EDDL can describe device functions, interactions supported by a device, device-supported objects, and other properties.

EDDL is used by a device vendor to create an electronic device definition (EDD) file that corresponds to a particular device. An EDD file is an operating system and automation system independent structured ASCII text file that describes the capabilities of a device to allow integration of the device with a host DCS system. This independence enables vendors to describe their devices in a manner that enables vendor independent interoperability of the device across host systems. EDD files describe device data, device vendor desired user interface characteristics, and device command handling, such as command ordering and timing.

Host DCSs provide tools to interpret EDD files in order to configure and handle the device, such as for monitoring or parameter handling, to support control applications.

EDDs are defined by device vendors and tested by the appropriate fieldbus supporting organization.

Figure 175 represents configuration using a DD file.

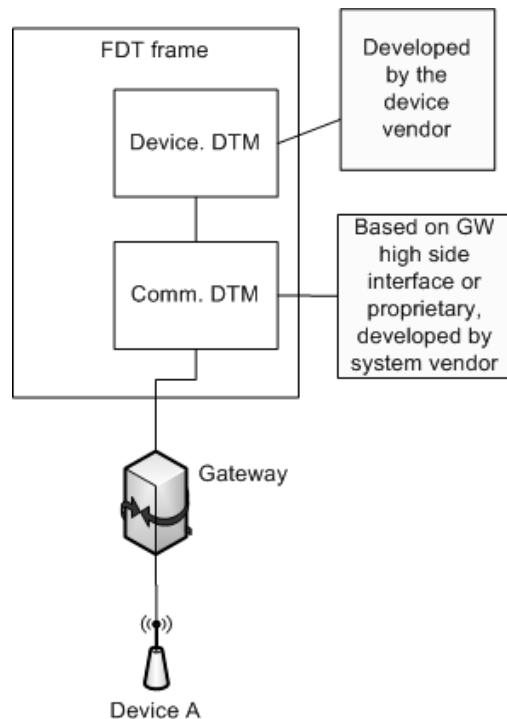


**Figure 175 – Configuration using an electronic device definition**

### R.3.3 Host configuration using field device tool/device type manager

The device functionality described by EDD is limited by the IEC standard. Additional device functionality (if any) that cannot be described via EDD can be supported via proprietary plug-ins or snap-ons. To provide this greater support, field device tool / device type manager (FDT/DTM) technology may be used. FDT/DTM technology requires, for example, FDT frame application support in the DCS. For further information on FDT/DTM, consult the FDT Group.

Figure 176 represents configuration using the FDT/DTM approach.



**Figure 176 – Configuration using FDT/DTM approach**

## R.4 Field device / distributed control systems integration

### R.4.1 General

Distributed control systems typically consist of devices such as controllers, human-machine-interface (HMI) stations, data historian servers, advanced applications, etc. HMI stations, historian servers, and advanced applications often employ interfaces with rich data semantics, such as OPC. Communication with controllers typically employs simpler protocols, such as Modbus, or Foundation Fieldbus High Speed Ethernet (FF-HSE).

### R.4.2 Foundation Fieldbus High Speed Ethernet

Application data integration with FF-HSE can, for example, be accomplished by mapping the native application data to FF transducer blocks. Application objects map to FF blocks, while object attributes map directly to the FF block parameters.

### R.4.3 Modbus

Application data can integrate with Modbus by assigning a Modbus address to the gateway. The gateway then may present a set of register tables to Modbus masters. Each object attribute may be mapped to a specific register. The host system may provide automated support for the mapping, or mapping may be performed manually by the user.

### R.4.4 Open connectivity for industrial automation

Open connectivity for industrial automation (OPC) allows client applications to access data in a consistent manner via an OPC server by referencing the data using a Tag.Parameter construct.

An OPC client may be supported by an OPC server in the host system or by a high-side OPC interface provided by a gateway to a standard-compliant system.

For example, this standard provides value, quality, and timestamp information in data publications, in support of OPC server access to online data. Native alarms and events also provide support for OPC client notification.

The OPC client may specify Tag.Parameter using the device name for the Tag, and a unique object name and attribute to represent the parameter (e.g., TI101.AITB1.PV). In the OPC server, the Tag is mapped to the device, the object instance maps to a particular object instance of a particular UAP, and the attribute name maps to the particular attribute identifier of the referenced object instance.

## R.5 Gateway

### R.5.1 General

Host system configuration of applications residing within the gateway itself, including data mapping (if necessary), is defined by the plant control network, which is the high side interface of the gateway that couples the WISN into a higher level control system. This includes, for example, configuration of a system management application or a tunneling application. Therefore, this annex describes in generalities the type of information that needs to be configured for gateway support.

### R.5.2 Devices supported

A host system configuration tool may need to establish the complement of standard-compliant devices with which the gateway will communicate.

### R.5.3 Data subscription

A host system configuration tool may need to establish the configuration of the dispersion objects in the gateway for the data the gateway will receive via publication.

### R.5.4 Data publication

A host system configuration tool may need to establish the configuration of the concentrator objects in the gateway for the data the gateway will itself publish.

### R.5.5 Client/server access

Non-management related client/server communications may, for example, be established by the gateway on an as-needed basis through interface objects.

### R.5.6 Alerts reception

A host system configuration tool may need to establish the alert categories associated with gateway-resident alert receiving object(s) (AROs).

## R.6 Asset management application support

### R.6.1 General

An asset management tool may access information about a device that is either stored in or accessed via the gateway by using plant control network services.

A gateway may access information directly from a field device to satisfy asset management requests. The gateway may, for example, employ client/server services to read data, to write data, or to execute a particular method on a particular object instance within the wireless device.

A gateway may act as a pass-through for asset information directly from an asset to an asset management application via a plant control network tunnel if the plant control network supports such a tunneling capability.

### R.6.2 Field device tool / device type manager

A DTM may be provided by a device vendor to provide process and device information to an asset management tool. A host system supporting an FDT frame can employ the device DTM and a communication DTM for the gateway to acquire the information necessary to manage the device via the gateway.

### R.6.3 HART

A standard-compliant device may be made to appear as a HART<sup>13</sup> native device on a HART asset management application (ASM) in several ways:

- Manually or using automation along with either explicitly coded or data-driven conversion rules provide a HART DD source file for the device. The HART DD file can be passed through a HART tokenizer to produce binary files representing the DD content. Most HART clients use the binary format of the DD files.
- Standard commands may be defined in HART to integrate ASM with this standard, such as a HART commands for READ\_ISA100\_ATTRIBUTE, WRITE\_ISA100\_ATTRIBUTE, and EXECUTE\_ISA100\_METHOD.
- Mapping tables in the gateway may be employed to define attribute value mapping that differs between this standard and HART, such as for engineering unit indices.

### R.6.4 OPC

Open connectivity for industrial automation (OPC) allows client applications to access data in a consistent manner via an OPC server. An OPC client may be supported by an OPC server in the host system or by a high-side OPC interface provided by a gateway to a standard-compliant system.

For example, device health information may be provided by the OPC server to an OPC client.

---

<sup>13</sup> HART is a registered trademark of HCF. This information is given for the convenience of users of the standard and does not constitute an endorsement of the trademark holders or any of their products. Compliance to this profile does not require use of the registered trademark. Use of the trademarks requires permission of the trade name holder.

**Annex S  
(INFORMATIVE)**

**Symmetric Key Operation Test Vectors**

**S.1 DPDU samples**

**[INGREDIENTS]**

- TsDur: 10464 [2^-20sec]
- DPDU Source EUI64: 0x00 00 00 00 00 00 00 01
- DL Key: 0xC0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
- TAI Time[TAINetworkTimeValue]: 0x00 01 02 03 04 05
- ACK Source EUI64: 0x00 00 00 00 00 00 00 02
- Channel: 0x02
- Sequence Number: 0x03
- DPDU Headers: 0x10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28
- DPDU Payload: 0x30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B
- DL ACK Headers: 0x10 11 12 13 14 15 16 17 18

**DPDU with DMIC32 Expected**

**[PRE-PROCESSED MATERIAL]**

- DPDU Nonce: 0x00 00 00 00 00 00 00 01 04 08 0C 10 14
- DPDU MIC: 0xBF 5A BB 7C
- DL ACK Nonce: 0x00 00 00 00 00 00 00 02 04 08 0C 10 14
- DL ACK authentication vector: 0x10 11 12 13 14 15 16 17 18 BF 5A BB 7C
- DL ACK MIC: 0xA7 5F 59 88

**[DELIVERABLE]**

- DPDU: 0x10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F 50 51 52 53 54 55 56 57 58 59 5A 5B BF 5A BB 7C
- DL ACK: 0x10 11 12 13 14 15 16 17 18 A7 5F 59 88

**DPDU with ENC-DMIC32 Expected**

**[PRE-PROCESSED MATERIAL]**

- DPDU Nonce: 0x00 00 00 00 00 00 00 01 04 08 0C 10 14
- DPDU MIC: 0xBF 5A BB 7C
- DL ACK Nonce: 0x00 00 00 00 00 00 00 02 04 08 0C 10 14
- DL ACK authentication vector: 0x10 11 12 13 14 15 16 17 18 BF 5A BB 7C
- Encrypted Payload: 0x23 F4 C4 3F BA 9B E4 3E D8 9B FD 36 A8 76 C7 99 27 14 E0 42 94 94 DE 64 B2 6B 14 18 51 9F 8D 11 36 F4 09 17 6B D6 A6 75 07 B1 D2 90
- DMIC: 0xD0 F6 B2 65

**[DERIVARIABLE]**

- DPDU: 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22 23 24 25 26 27 28 23  
F4 C4 3F BA 9B E4 3E D8 9B FD 36 A8 76 C7 99 27 14 E0 42 94 94 DE 64 B2 6B 14 18  
51 9F 8D 11 36 F4 09 17 6B D6 A6 75 07 B1 D2 90 D0 F6 B2 65
- DL ACK: 0x10 11 12 13 14 15 16 17 18 A7 5F 59 88

**S.2 TPDU samples****[INGREDIENTS]**

- TPDU time creation[TAINetworkTimeValue]: 0x00 01 02 03 04 05
- Key: 0xC0 C1 C2 C3 C4 C5 C6 C7 C8 C9 CA CB CC CD CE CF
- Crypto Key Identifier Mode: 0x00
- Crypto Key Identifier = 0x10
- Source EUI64: 0x00 00 00 00 00 00 00 01
- Source IPv6: 0xFE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 01
- Dest IPv6: 0xFE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 02
- Source Port: 0x00 01
- Dest port: 0x00 02
- Application Layer Payload: 0x10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F 20 21 22  
23 24 25 26 27 28 29 2A 2B
- Application Layer Payload Length: 0x1C(28)

**TPDU with ENC-TMIC-32 Expected:****[PRE-PROCESSED MATERIAL]**

- TPDU Pseudo header: 0xFE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 01 FE 80 00 00  
00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 2B 00 11 00 01 00 02
- TPDU Nonce: 0x00 00 00 00 00 00 00 01 04 08 0C 10 FF
- Security header: 0xA0 0C 10

**[DELIVERABLE]**

- TPDU: 0x 00 01 00 02 00 23 00 00 A0 18 20 8E 7D 08 B8 8C CC 16 7F 56 CF 72 19 13  
B6 06 FF FD 6B F2 C2 9A 04 BA FC E9 5E 15 C8 44 4C 0A 43 88

**TPDU with TMIC-32 Expected:****[PRE-PROCESSED MATERIAL]**

- TPDU Pseudo header: 0xFE 80 00 00 00 00 00 00 00 00 00 00 00 00 00 01 FE 80 00 00  
00 00 00 00 00 00 00 00 00 00 00 00 00 02 00 2B 00 11 00 01 00 02
- TPDU Nonce: 0x00 00 00 00 00 00 00 01 04 08 0C 10 FF
- Security header: 0x20 0C 10

**[DELIVERABLE]**

- TPDU: 0x 00 01 00 02 00 23 00 00 20 0C 10 0x10 11 12 13 14 15 16 17 18 19 1A 1B 1C  
1D 1E 1F 20 21 22 23 24 25 26 27 28 29 2A 2B 7E 8C 35 57

## Annex T (INFORMATIVE)

### Data link header and network header for join requests

#### T.1 Overview

This annex illustrates the DL header and NL header for a typical join request.

#### T.2 MAC header (MHR)

MAC header for join messages is shown in Table 526. IEEE convention shows bit 0 on the right, which is the order of transmission. Table 526 follows the convention of this standard, showing bit 7 on the left regardless of the order of transmission.

**Table 526 – Sample MHR for join request**

	octets	bits																
		7	6	5	4	3	2	1	0									
Frame Control	2 Octets  Frame control	Reserved=0	PAN ID Compress =1 (yes)	ACK Request = 0 (no)	Frame Pending =0 (no)	Security Enabled =0 (no)	Frame Type =1 (Data)											
		Source Addressing Mode =3 (64-bit)	Frame Version =1 (2006)		Dest Addressing Mode =2 (16-bit)	Reserved=0												
Sequence Number	1 octet	(variable)																
Addressing Fields	2 octets	PAN ID (LSB), from advertisement																
	2 octets	Destination Address (16-bit, LSB), from advertisement																
	8 octets	Source Address (64-bit, LSB), device's EUI-64																

#### T.3 DL header (DHR)

DL header for join messages is shown in Table 527. This example assumes that the advertisement does not specify slow hopping.

**Table 527 – Sample DHR for join request**

Sub header	octets	bits																					
		7	6	5	4	3	2	1	0														
DHDR	1 octet	ACK needed =1 (yes)	Signal quality in ACK =0 (no)	Request EUI-64 =0 (no)	Include DAUX = 0 (no)	Include slow hopping offset =0 (no)	Clock recipient =1 (yes)	DL version = 00															
DMXHR	1 octet	Reserved=0			Key identifier mode =1		Security level =1 (MIC-32)																
	1 octet	Crypto key identifier = 0 (K_Global)																					
DAUX	0 octet	(absent by DHDR setting)																					
DROUT	1 octet	Compress=1	Priority =0 (Doesn't matter)			DIForwardLimit =1																	
	1 octet	GraphID (Unsigned8) =0 (Single hop source routing)																					
DADDR	1 octet	DE=0	LH=0	ECN=0	Reserved=0																		
	1 octet	SrcAddr = 0 (Use 64-bit address in MHR)																					
	1 octet	DestAddr = 0 (Use 16-bit address in MHR)																					

#### T.4 NL header

Network header for join messages is shown in Table 528.

**Table 528 – Network header for join messages**

## Bibliography

D. R. L. Brown, R. P. Gallant, S. A. Vanstone, *Provably secure implicit certificate schemes*, in Financial cryptography 2001, K. Nyberg, H. Heys, Eds. Lecture notes in Computer Science, Vol 2339, pp. 156-165, Berlin: Springer, 2001.

ERC/REC 70-03, *Relating to the use of short range devices (SRD), Annex 1, Band E*

ETSI EN 300 220-1, *Electromagnetic compatibility and radio spectrum matters (ERM) – Short range devices – Technical characteristics and test methods for radio equipment to be used in the 25 MHz to 1 000 MHz frequency range with power levels ranging up to 500 mW – Part 1: Parameters intended for regulatory purposes*

ETSI EN 300 328-1, *Radio equipment and systems (RES) – Wideband transmission systems – Technical characteristics and test conditions for data transmission equipment operating in the 2,4 GHz ISM band and using spread spectrum modulation techniques*

ETSI EN 300 328-2, *Electromagnetic compatibility and radio spectrum matters (ERM) – Wideband Transmission systems – Data transmission equipment operating in the 2,4 GHz ISM band and using spread spectrum modulation techniques – Part 2: Harmonized EN covering essential requirements under article 3.2 of the R&TTE Directive*

R. Housley, D. Whiting, N. Ferguson, *Counter with CBC-MAC (CCM)*, submitted to NIST., June 3, 2002

IEC 61158 family, *Digital data communications for measurement and control – Fieldbus for use in industrial control systems*

IEEE 754, *IEEE standard for binary floating-point arithmetic*

IEEE 802.11, *IEEE standards for information technology – Telecommunications and information exchange between systems – Local and metropolitan area network – Specific requirements – Part 11: Wireless LAN medium access control (MAC) and physical layer (PhL) specifications*

IEEE 802.16, *IEEE Standard for local and metropolitan area networks–Part 16: Air interface for fixed broadband wireless access systems*

IERS conventions: IERS technical note 32

IETF RFC 3280, *Internet X.509 Public key infrastructure certificate and certificate revocation list (CRL) profile*, Internet request for comments 3280, R. Housley, W. Polk, W. Ford, D. Solo, April 2002.

ISA TR100.00.01:2006, *The automation engineer's guide to wireless technology–Part 1: The physics of radio, a tutorial*

ISC RSS 210, *Radio standards specification 210 – Low-power license-exempt radio communication devices (all frequency bands): Category I equipment*

J. Jonsson, *On the security of CTR + CBC-MAC*, in *Proceedings of selected areas in cryptography – SAC 2002*, K. Nyberg, H. Heys, Eds. Lecture notes in computer science, Vol. 2595, pp. 76-93, Berlin: Springer, 2002.

J. Jonsson, *On the security of CTR + CBC-MAC*, NIST mode of operation – Additional CCM documentation

A.J. Menezes, P.C. van Oorschot, S.A. Vanstone and process control industry data structures are both, *Handbook of applied cryptography*, Boca Raton: CRC Press, 1997.

NAMUR Recommendation 105, *Specifications for integrating fieldbus devices*

NAMUR Recommendation 107, *Self-monitoring and diagnostics of field devices*

NIST SP 800-22, *A statistical test suite for random and pseudorandom number generators for cryptographic applications*

NIST SP800-88, *Guidelines for media sanitization*

PKIX, L. Bassham, R. Housley, W. Polk, *Algorithms and identifiers for the internet X.509 Public key infrastructure certificate and CRL profile*, Internet draft, PKIX Working Group, October 2001.

P. Rogaway, D. Wagner, *A critique of CCM*, IACR ePrint Archive 2003-070, April 13, 2003.

F. Stajano, *The resurrecting duckling: What next?*, in *Proceedings of the 8th international workshop on security protocols*, B. Crispo, M. Roe, and B. Crispo, Eds., Lecture notes in computer science, Vol. 2133, Berlin: Springer-Verlag, April 2000.

F. Stajano, R. Anderson, *The resurrecting duckling: Security issues in ad-hoc wireless networks*, in *Proceedings of the 7th international workshop on security protocols*, B. Christianson, B. Crispo, J.A. Malcolm, and M. Roe, Eds., Lecture notes in computer science, Vol. 1796, Berlin: Springer-Verlag, 1999.

US Code of Federal Regulations (CFR) Title 47, Chapter I, Part 15, – *Telecommunication – Part 15: Radio frequency devices*

---

This page intentionally left blank.



Developing and promulgating sound consensus standards, recommended practices, and technical reports is one of ISA's primary goals. To achieve this goal the Standards and Practices Department relies on the technical expertise and efforts of volunteer committee members, chairmen and reviewers.

ISA is an American National Standards Institute (ANSI) accredited organization. ISA administers United States Technical Advisory Groups (USTAGs) and provides secretariat support for International Electrotechnical Commission (IEC) and International Organization for Standardization (ISO) committees that develop process measurement and control standards. To obtain additional information on the Society's standards program, please write:

ISA  
Attn: Standards Department  
67 Alexander Drive  
P.O. Box 12277  
Research Triangle Park, NC 27709

ISBN: 978-1-936007-96-7