

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

**Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 6-2: Spécification de protocole de la couche application – Éléments de
Type 2**



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2010 IEC, Geneva, Switzerland

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either IEC or IEC's member National Committee in the country of the requester.

If you have any questions about IEC copyright or have an enquiry about obtaining additional rights to this publication, please contact the address below or your local IEC member National Committee for further information.

Droits de reproduction réservés. Sauf indication contraire, aucune partie de cette publication ne peut être reproduite ni utilisée sous quelque forme que ce soit et par aucun procédé, électronique ou mécanique, y compris la photocopie et les microfilms, sans l'accord écrit de la CEI ou du Comité national de la CEI du pays du demandeur.

Si vous avez des questions sur le copyright de la CEI ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de la CEI de votre pays de résidence.

IEC Central Office
3, rue de Varembe
CH-1211 Geneva 20
Switzerland

Tel.: +41 22 919 02 11
Fax: +41 22 919 03 00
info@iec.ch
www.iec.ch

About the IEC

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

About IEC publications

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

Useful links:

IEC publications search - www.iec.ch/searchpub

The advanced search enables you to find IEC publications by a variety of criteria (reference number, text, technical committee,...).

It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

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

Electropedia - www.electropedia.org

The world's leading online dictionary of electronic and electrical terms containing more than 30 000 terms and definitions in English and French, with equivalent terms in additional languages. Also known as the International Electrotechnical Vocabulary (IEV) on-line.

Customer Service Centre - webstore.iec.ch/csc

If you wish to give us your feedback on this publication or need further assistance, please contact the Customer Service Centre: csc@iec.ch.

A propos de la CEI

La Commission Electrotechnique Internationale (CEI) est la première organisation mondiale qui élabore et publie des Normes internationales pour tout ce qui a trait à l'électricité, à l'électronique et aux technologies apparentées.

A propos des publications CEI

Le contenu technique des publications de la CEI est constamment revu. Veuillez vous assurer que vous possédez l'édition la plus récente, un corrigendum ou amendement peut avoir été publié.

Liens utiles:

Recherche de publications CEI - www.iec.ch/searchpub

La recherche avancée vous permet de trouver des publications CEI en utilisant différents critères (numéro de référence, texte, comité d'études,...).

Elle donne aussi des informations sur les projets et les publications remplacées ou retirées.

Just Published CEI - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications de la CEI. Just Published détaille les nouvelles publications parues. Disponible en ligne et aussi une fois par mois par email.

Electropedia - www.electropedia.org

Le premier dictionnaire en ligne au monde de termes électroniques et électriques. Il contient plus de 30 000 termes et définitions en anglais et en français, ainsi que les termes équivalents dans les langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (VEI) en ligne.

Service Clients - webstore.iec.ch/csc

Si vous désirez nous donner des commentaires sur cette publication ou si vous avez des questions contactez-nous: csc@iec.ch.

INTERNATIONAL STANDARD

NORME INTERNATIONALE

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

**Réseaux de communication industriels – Spécifications des bus de terrain –
Partie 6-2: Spécification de protocole de la couche application – Eléments de
Type 2**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

PRICE CODE
CODE PRIX

XH

ICS 25.040.40; 35.100.70; 35.110

ISBN 978-2-83220-125-1

**Warning! Make sure that you obtained this publication from an authorized distributor.
Attention! Veuillez vous assurer que vous avez obtenu cette publication via un distributeur agréé.**

CONTENTS

FOREWORD.....	12
INTRODUCTION.....	14
1 Scope.....	15
1.1 General.....	15
1.2 Specifications.....	15
1.3 Conformance.....	16
2 Normative references	16
3 Terms, definitions, symbols, abbreviations and conventions	17
3.1 Terms and definitions from other ISO/IEC standards	17
3.2 Terms and definitions from IEC 61158-5-2.....	18
3.3 Additional terms and definitions.....	18
3.4 Abbreviations and symbols.....	24
3.5 Conventions	24
3.6 Conventions used in state machines	29
4 Abstract syntax.....	30
4.1 FAL PDU abstract syntax	30
4.2 Data abstract syntax specification	120
4.3 Encapsulation abstract syntax.....	124
5 Transfer syntax	138
5.1 Compact encoding.....	138
5.2 Data type reporting.....	145
6 Structure of FAL protocol state machines	150
7 AP-Context state machine.....	150
7.1 Overview.....	150
7.2 Connection object state machine.....	150
8 FAL service protocol machine (FSPM).....	160
8.1 General.....	160
8.2 Primitive definitions.....	160
8.3 Parameters of primitives.....	164
8.4 FSPM state machines.....	165
9 Application relationship protocol machines (ARPMs)	166
9.1 General.....	166
9.2 Connection-less ARPM (UCMM).....	166
9.3 Connection-oriented ARPMs (transports).....	176
10 DLL mapping protocol machine 1 (DMPM 1).....	247
10.1 General.....	247
10.2 Link producer	247
10.3 Link consumer.....	248
10.4 Primitive definitions	248
10.5 DMPM state machine	250
10.6 Data-link Layer service selection.....	251
11 DLL mapping protocol machine 2 (DMPM 2).....	251
11.1 General.....	251
11.2 Mapping of UCMM PDUs.....	252
11.3 Mapping of transport class 0 and class 1 PDUs.....	257

11.4 Mapping of transport class 2 and class 3 PDU's	258
11.5 Mapping of transport classes 4 to 6	258
11.6 IGMP Usage	258
11.7 Quality of Service (QoS) for CP 2/2 messages	259
11.8 Management of an encapsulation session	262
12 DLL mapping protocol machine 3 (DMPM 3)	263
Bibliography	264
Figure 1 – Attribute table format and terms	25
Figure 2 – Service request/response parameter	25
Figure 3 – Example of an STD	29
Figure 4 – Network connection parameters	49
Figure 5 – Time tick	51
Figure 6 – Connection establishment time-out	53
Figure 7 – Transport Class Trigger attribute	86
Figure 8 – CP2/3_initial_comm_characteristics attribute format	89
Figure 9 – Segment type	97
Figure 10 – Port segment	98
Figure 11 – Logical segment encoding	100
Figure 12 – Extended network segment	104
Figure 13 – Encapsulation message	124
Figure 14 – FixedLengthBitString compact encoding bit placement rules	142
Figure 15 – Example compact encoding of a SWORD FixedLengthBitString	142
Figure 16 – Example compact encoding of a WORD FixedLengthBitString	142
Figure 17 – Example compact encoding of a DWORD FixedLengthBitString	143
Figure 18 – Example compact encoding of a LWORD FixedLengthBitString	143
Figure 19 – Example 2 of formal encoding of a structure type specification	147
Figure 20 – Example of abbreviated encoding of a structure type specification	148
Figure 21 – Example 1 of formal encoding of an array type specification	148
Figure 22 – Example 2 of formal encoding of an array type specification	149
Figure 23 – Example 1 of abbreviated encoding of an array type specification	149
Figure 24 – Example 2 of abbreviated encoding of an array type specification	150
Figure 25 – I/O Connection object state transition diagram	151
Figure 26 – Bridged Connection object state transition diagram	155
Figure 27 – Explicit Messaging Connection object state transition diagram	157
Figure 28 – State transition diagram of UCMM client	169
Figure 29 – State transition diagram of high-end UCMM server	170
Figure 30 – State transition diagram of low-end UCMM server	172
Figure 31 – Sequence diagram for a UCMM with one outstanding message	174
Figure 32 – Sequence diagram for a UCMM with multiple outstanding messages	175
Figure 33 – TPDU buffer	176
Figure 34 – Data flow diagram using a client transport class 0 and server transport class 0	179
Figure 35 – Sequence diagram of data transfer using transport class 0	179

Figure 36 – Class 0 client STD	180
Figure 37 – Class 0 server STD	181
Figure 38 – Data flow diagram using client transport class 1 and server transport class 1	182
Figure 39 – Sequence diagram of data transfer using client transport class 1 and server transport class 1	183
Figure 40 – Class 1 client STD	185
Figure 41 – Class 1 server STD	186
Figure 42 – Data flow diagram using client transport class 2 and server transport class 2	188
Figure 43 – Diagram of data transfer using client transport class 2 and server transport class 2 <i>without</i> returned data	189
Figure 44 – Sequence diagram of data transfer using client transport class 2 and server transport class 2 <i>with</i> returned data	190
Figure 45 – Class 2 client STD	191
Figure 46 – Class 2 server STD	193
Figure 47 – Data flow diagram using client transport class 3 and server transport class 3	196
Figure 48 – Sequence diagram of data transfer using client transport class 3 and server transport class 3 <i>without</i> returned data	197
Figure 49 – Sequence diagram of data transfer using client transport class 3 and server transport class 3 <i>with</i> returned data	198
Figure 50 – Class 3 client STD	200
Figure 51 – Class 3 server STD	203
Figure 52 – Data flow diagram using transport classes 4 and 5	205
Figure 53 – Sequence diagram of message exchange using transport classes 4 and 5	206
Figure 54 – Sequence diagram of messages overwriting each other	207
Figure 55 – Sequence diagram of queued message exchange using transport classes 4 and 5	208
Figure 56 – Sequence diagram of retries using transport classes 4 and 5	208
Figure 57 – Sequence diagram of idle traffic using transport classes 4 and 5	209
Figure 58 – Classes 4 and 5 basic structure	210
Figure 59 – Class 6 basic structure	211
Figure 60 – Classes 4 to 6 general STD	212
Figure 61 – Class 4 sender STD	214
Figure 62 – Class 4 receiver STD	217
Figure 63 – Sequence diagram of three fragments using transport class 5	220
Figure 64 – Sequence diagram of fragmentation with retries using transport class 5	221
Figure 65 – Sequence diagram of two fragments using transport class 5	221
Figure 66 – Sequence diagram of aborted message using transport class 5	222
Figure 67 – Class 5 sender STD	223
Figure 68 – Class 5 receiver STD	226
Figure 69 – Data flow diagram for transport class 6	231
Figure 70 – Sequence diagram of message exchange using transport class 6	233
Figure 71 – Sequence diagram of retries using transport class 6	233
Figure 72 – Sequence diagram of idle traffic using transport class 6	234

Figure 73 – Sequence diagram of request overwriting null	235
Figure 74 – Sequence diagram of response overwriting ACK of null	235
Figure 75 – Sequence diagram of three fragments using transport class 6	236
Figure 76 – Sequence diagram of fragmentation with retries using transport class 6	237
Figure 77 – Sequence diagram of two fragments using transport class 6	237
Figure 78 – Sequence diagram of aborted fragmented sequence using transport class 6	238
Figure 79 – Class 6 client STD	239
Figure 80 – Class 6 server STD	242
Figure 81 – Data flow diagram for a link producer and consumer	247
Figure 82 – State transition diagram for a link producer	250
Figure 83 – State transition diagram for a link consumer	251
Figure 84 – DS field in the IP header	261
Figure 85 – IEEE 802.1Q tagged frame	261
Table 1 – Get_Attribute_All response service rules	26
Table 2 – Example class level object/service specific response data of Get_Attribute_All	26
Table 3 – Example Get_Attribute_All data array method	27
Table 4 – Set_Attribute_All request service rules	28
Table 5 – Example Set_Attribute_All attribute ordering method	28
Table 6 – Example Set_Attribute_All data array method	28
Table 7 – State event matrix format	30
Table 8 – Example state event matrix	30
Table 9 – UCMM_PDU header format	33
Table 10 – UCMM command codes	33
Table 11 – Transport class 0 header	34
Table 12 – Transport class 1 header	35
Table 13 – Transport class 2 header	35
Table 14 – Transport class 3 header	35
Table 15 – Classes 4 to 6 header format	35
Table 16 – Real-time data header – exclusive owner	36
Table 17 – Real-time data header – redundant owner	37
Table 18 – Forward_Open request format	39
Table 19 – Forward_Open_Good response format	40
Table 20 – Forward_Open_Bad response format	40
Table 21 – Large_Forward_Open request format	41
Table 22 – Large_Forward_Open_Good response format	42
Table 23 – Large_Forward_Open_Bad response format	42
Table 24 – Forward_Close request format	43
Table 25 – Forward_Close_Good response format	43
Table 26 – Forward_Close_Bad response format	44
Table 27 – Unconnected_Send request format	45
Table 28 – Unconnected_Send_Good response format	45

Table 29 – Unconnected_Send_Bad response format	46
Table 30 – Get_Connection_Data request format	46
Table 31 – Get_Connection_Data response format	47
Table 32 – Search_Connection_Data request format	47
Table 33 – Get_Object_Owner request format	48
Table 34 – Forward_Open_Good response format	48
Table 35 – Time-out multiplier	51
Table 36 – Time tick units	51
Table 37 – Selection of connection ID	55
Table 38 – Transport class, trigger and Is_Server format	55
Table 39 – MR_Request_Header format	56
Table 40 – MR_Response_Header format	56
Table 41 – Structure of Get_Attribute_All_ResponsePDU body	57
Table 42 – Structure of Set_Attribute_All_RequestPDU body	57
Table 43 – Structure of Get_Attribute_List_RequestPDU body	57
Table 44 – Structure of Get_Attribute_List_ResponsePDU body	57
Table 45 – Structure of Set_Attribute_List_RequestPDU body	57
Table 46 – Structure of Set_Attribute_List_ResponsePDU body	58
Table 47 – Structure of Reset_RequestPDU body	58
Table 48 – Structure of Reset_ResponsePDU body	58
Table 49 – Structure of Start_RequestPDU body	58
Table 50 – Structure of Start_ResponsePDU body	58
Table 51 – Structure of Stop_RequestPDU body	59
Table 52 – Structure of Stop_ResponsePDU body	59
Table 53 – Structure of Create_RequestPDU body	59
Table 54 – Structure of Create_ResponsePDU body	59
Table 55 – Structure of Delete_RequestPDU body	59
Table 56 – Structure of Delete_ResponsePDU body	59
Table 57 – Structure of Get_Attribute_Single_ResponsePDU body	60
Table 58 – Structure of Set_Attribute_Single_RequestPDU body	60
Table 59 – Structure of Set_Attribute_Single_ResponsePDU body	60
Table 60 – Structure of Find_Next_Object_Instance_RequestPDU body	60
Table 61 – Structure of Find_Next_Object_Instance_ResponsePDU body	60
Table 62 – Structure of Apply_Attributes_RequestPDU body	61
Table 63 – Structure of Apply_Attributes_ResponsePDU body	61
Table 64 – Structure of Save_RequestPDU body	61
Table 65 – Structure of Save_ResponsePDU body	61
Table 66 – Structure of Restore_RequestPDU body	61
Table 67 – Structure of Restore_ResponsePDU body	61
Table 68 – Structure of Group_Sync_RequestPDU body	62
Table 69 – Structure of Group_Sync_ResponsePDU body	62
Table 70 – Identity object class attributes	62
Table 71 – Identity object instance attributes	62

Table 72 – Identity object bit definitions for status instance attribute	63
Table 73 – Bits 4 – 7 of status instance attribute	64
Table 74 – Class level object/service specific response data of Get_Attribute_All	64
Table 75 – Instance level object/service specific response data of Get_Attribute_All	65
Table 76 – Modified instance level object/service specific response data of Get_Attribute_All	65
Table 77 – Object-specific parameter for Reset	66
Table 78 – Message Router object class attributes	66
Table 79 – Message Router object instance attributes	66
Table 80 – Class level object/service specific response data of Get_Attribute_All	67
Table 81 – Instance level object/service specific response data of Get_Attribute_All	67
Table 82 – Assembly object class attributes	67
Table 83 – Assembly object instance attributes	67
Table 84 – Acknowledge Handler object class attributes	68
Table 85 – Acknowledge Handler object instance attributes	69
Table 86 – Structure of Add_AckData_Path_RequestPDU body	69
Table 87 – Structure of Remove_AckData_Path_RequestPDU body	69
Table 88 – Time Sync object instance attributes	70
Table 89 – ClockIdentity encoding for different network implementations	73
Table 90 – ClockClass values	73
Table 91 – TimeAccuracy values	74
Table 92 – TimePropertyFlags bit values	74
Table 93 – TimeSource values	75
Table 94 – Types of Clock	75
Table 95 – Parameter object class attributes	75
Table 96 – Parameter Class Descriptor bit values	76
Table 97 – Parameter object instance attributes	77
Table 98 – Semantics of Descriptor Instance attribute	77
Table 99 – Minimum and Maximum Value semantics	79
Table 100 – Scaling Formula attributes	80
Table 101 – Scaling links	80
Table 102 – Structure of Get_Enum_String_RequestPDU body	81
Table 103 – Structure of Get_Enum_String_ResponsePDU body	81
Table 104 – Enumerated strings Type versus Parameter data type	81
Table 105 – Connection Manager object class attributes	82
Table 106 – Connection Manager object instance attributes	82
Table 107 – Connection object class attributes	83
Table 108 – Connection object instance attributes	84
Table 109 – Values assigned to the state attribute	85
Table 110 – Values assigned to the instance_type attribute	85
Table 111 – Possible values within Direction Bit	86
Table 112 – Possible values within Production Trigger Bits	86
Table 113 – Possible values within Transport Class Bits	87

Table 114 – Transport Class_Trigger attribute	88
Table 115 – Values defined for the CP2/3_produced_connection_id attribute	88
Table 116 – Values defined for the CP2/3_consumed_connection_id attribute	89
Table 117 – Values for the Initial Production Characteristics nibble	90
Table 118 – Values for the Initial Consumption Characteristics nibble.....	91
Table 119 – Values for the watchdog_timeout_action.....	94
Table 120 – Structure of Connection_Bind_RequestPDU body.....	95
Table 121 – Object specific status for Connection_Bind service.....	96
Table 122 – Structure of Producing_Application_Lookup_RequestPDU body	96
Table 123 – Structure of Producing_Application_Lookup_ResponsePDU body	96
Table 124 – Producing_Application_Lookup Service status codes	96
Table 125 – Possible port segment examples	98
Table 126 – TCP/IP link address examples	99
Table 127 – Electronic key segment format.....	101
Table 128 – Logical segments examples.....	102
Table 129 – Network segments	102
Table 130 – Extended subtype definitions	104
Table 131 – Data segment	104
Table 132 – ANSI_Extended_Symbol segment	105
Table 133 – Addressing categories	105
Table 134 – Class code ID ranges	106
Table 135 – Attribute ID ranges	106
Table 136 – Service code ranges	106
Table 137 – Class codes	107
Table 138 – Reserved class attributes for all object class definitions	107
Table 139 – Common services list	108
Table 140 – Acknowledge Handler object specific services list	109
Table 141 – Parameter object specific services list	109
Table 142 – Services specific to Connection Manager	109
Table 143 – Services specific to Connection object.....	109
Table 144 – Device type numbering	110
Table 145 – Connection Manager service request error codes	111
Table 146 – General status codes.....	116
Table 147 – Identity object status codes	118
Table 148 – Encapsulation header	125
Table 149 – Encapsulation command codes	125
Table 150 – Encapsulation status codes	126
Table 151 – Options flags	127
Table 152 – Nop request encapsulation header	127
Table 153 – RegisterSession request encapsulation header	127
Table 154 – RegisterSession request data portion	128
Table 155 – Options flags	128
Table 156 – RegisterSession reply encapsulation header	128

Table 157 – RegisterSession reply data portion	128
Table 158 – UnRegisterSession request encapsulation header	129
Table 159 – ListServices request encapsulation header	129
Table 160 – ListServices reply encapsulation header	130
Table 161 – ListServices reply data portion	130
Table 162 – Service type codes	130
Table 163 – Communications capability flags	130
Table 164 – ListIdentity request encapsulation header	131
Table 165 – ListIdentity reply encapsulation header	131
Table 166 – ListIdentity reply data portion	132
Table 167 – ListInterfaces request encapsulation header	132
Table 168 – ListInterfaces reply encapsulation header	133
Table 169 – SendRRData request encapsulation header	133
Table 170 – SendRRData request data portion	133
Table 171 – SendRRData reply encapsulation header	134
Table 172 – SendUnitData request encapsulation header	134
Table 173 – SendUnitData request data portion	135
Table 174 – Common packet format	135
Table 175 – Address and data item structure	135
Table 176 – Address type ID's	135
Table 177 – Data type ID's	136
Table 178 – Null address type	136
Table 179 – Connected address type	136
Table 180 – Sequenced address type	137
Table 181 – UCMM data type	137
Table 182 – Connected data type	137
Table 183 – Sockaddr info items	137
Table 184 – BOOLEAN encoding	139
Table 185 – Example compact encoding of a BOOL value	139
Table 186 – Encoding of SignedInteger values	139
Table 187 – Example compact encoding of a SignedInteger value	139
Table 188 – UnsignedInteger values	139
Table 189 – Example compact encoding of an UnsignedInteger	140
Table 190 – FixedLengthReal values	140
Table 191 – Example compact encoding of a REAL value	140
Table 192 – Example compact encoding of a LREAL value	140
Table 193 – FixedLengthReal values	140
Table 194 – STRING value	141
Table 195 – STRING2 value	141
Table 196 – STRINGN value	141
Table 197 – SHORT_STRING value	141
Table 198 – Example compact encoding of a STRING value	141
Table 199 – Example compact encoding of STRING2 value	142

Table 200 – SHORT_STRING type	142
Table 201 – Example compact encoding of a single dimensional ARRAY	143
Table 202 – Example compact encoding of a multi-dimensional ARRAY	144
Table 203 – Example compact encoding of a STRUCTURE	144
Table 204 – Identification codes and descriptions of elementary data types	146
Table 205 – Example 1 of formal encoding of a structure type specification	147
Table 206 – I/O Connection state event matrix.....	151
Table 207 – Bridged Connection state event matrix	156
Table 208 – Explicit Messaging Connection state event matrix	158
Table 209 – Primitives issued by FAL user to FSPM	161
Table 210 – Primitives issued by FAL user to FSPM	162
Table 211 – Primitives issued by FSPM to FAL user	164
Table 212 – Parameters used with primitives exchanged between FAL user and FSPM	165
Table 213 – Primitives issued by FSPM to ARPM	167
Table 214 – Primitives issued by ARPM to FSPM	167
Table 215 – Parameters used with primitives exchanged between FSPM and ARPM	168
Table 216 – UCMM client states	168
Table 217 – State event matrix of UCMM client.....	169
Table 218 – High-end UCMM server states.....	170
Table 219 – State event matrix of high-end UCMM server.....	171
Table 220 – Low-end UCMM server states.....	172
Table 221 – State event matrix of low-end UCMM server	173
Table 222 – Notification	176
Table 223 – Transport classes	177
Table 224 – Primitives issued by FSPM to ARPM	177
Table 225 – Primitives issued by ARPM to FSPM	178
Table 226 – Parameters used with primitives exchanged between FSPM and ARPM	178
Table 227 – Class 0 transport client states	180
Table 228 – Class 0 client SEM	180
Table 229 – Class 0 transport server states	181
Table 230 – Class 0 server SEM.....	181
Table 231 – Class 1 transport client states	184
Table 232 – Class 1 client SEM	185
Table 233 – Class 1 transport server states	186
Table 234 – Class 1 server SEM.....	187
Table 235 – Class 2 transport client states	191
Table 236 – Class 2 client SEM	192
Table 237 – Class 2 transport server states	193
Table 238 – Class 2 server SEM.....	194
Table 239 – Class 3 transport client states	199
Table 240 – Class 3 client SEM	200
Table 241 – Class 3 transport server states	202
Table 242 – Class 3 server SEM.....	204

Table 243 – Write and trigger events in class 4 and 5 transport	206
Table 244 – Common states for transport classes 4 to 6	211
Table 245 – Classes 4 to 6 general SEM	212
Table 246 – Class 4 transport sender states	214
Table 247 – Class 4 sender SEM	215
Table 248 – Class 4 transport receiver states	216
Table 249 – Class 4 receiver SEM	218
Table 250 – Class 5 transport sender states	223
Table 251 – Class 5 sender SEM	224
Table 252 – Class 5 transport receiver states	226
Table 253 – Class 5 receiver SEM	228
Table 254 – Class 6 transport client states	239
Table 255 – Class 6 client state event matrix	240
Table 256 – Class 6 transport server states	241
Table 257 – Class 6 server SEM	244
Table 258 – Primitives issued by ARPM to DMPM	248
Table 259 – Primitives issued by DMPM to ARPM	248
Table 260 – Parameters used with primitives exchanged between ARPM and DMPM	248
Table 261 – Primitives exchanged between data-link layer and DMPM	249
Table 262 – Parameters used with primitives exchanged between DMPM and Data-link	249
Table 263 – Link producer states	250
Table 264 – State event matrix of link producer	250
Table 265 – Link consumer states	251
Table 266 – State event matrix of link consumer	251
Table 267 – Network Connection ID selection	253
Table 268 – Example multicast assignments	256
Table 269 – UDP data format for class 0 and class 1	257
Table 270 – Default DSCP and IEEE 802.1D mapping	261

INTERNATIONAL ELECTROTECHNICAL COMMISSION

**INDUSTRIAL COMMUNICATION NETWORKS –
FIELDBUS SPECIFICATIONS –****Part 6-2: Application layer protocol specification –
Type 2 elements**

FOREWORD

- 1) The International Electrotechnical Commission (IEC) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, IEC publishes International Standards, Technical Specifications, Technical Reports, Publicly Available Specifications (PAS) and Guides (hereafter referred to as “IEC Publication(s)”). Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
- 2) The formal decisions or agreements of IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested IEC National Committees.
- 3) IEC Publications have the form of recommendations for international use and are accepted by IEC National Committees in that sense. While all reasonable efforts are made to ensure that the technical content of IEC Publications is accurate, IEC cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.
- 4) In order to promote international uniformity, IEC National Committees undertake to apply IEC Publications transparently to the maximum extent possible in their national and regional publications. Any divergence between any IEC Publication and the corresponding national or regional publication shall be clearly indicated in the latter.
- 5) IEC itself does not provide any attestation of conformity. Independent certification bodies provide conformity assessment services and, in some areas, access to IEC marks of conformity. IEC is not responsible for any services carried out by independent certification bodies.
- 6) All users should ensure that they have the latest edition of this publication.
- 7) No liability shall attach to IEC or its directors, employees, servants or agents including individual experts and members of its technical committees and IEC National Committees for any personal injury, property damage or other damage of any nature whatsoever, whether direct or indirect, or for costs (including legal fees) and expenses arising out of the publication, use of, or reliance upon, this IEC Publication or any other IEC Publications.
- 8) Attention is drawn to the Normative references cited in this publication. Use of the referenced publications is indispensable for the correct application of this publication.
- 9) Attention is drawn to the possibility that some of the elements of this IEC Publication may be the subject of patent rights. IEC shall not be held responsible for identifying any or all such patent rights.

NOTE 1 Use of some of the associated protocol types is restricted by their intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a particular data-link layer protocol type to be used with physical layer and application layer protocols in Type combinations as specified explicitly in the profile parts. Use of the various protocol types in other combinations may require permission from their respective intellectual-property-right holders.

International Standard IEC 61158-6-2 has been prepared by subcommittee 65C: Industrial networks, of IEC technical committee 65: Industrial-process measurement, control and automation.

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

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

- Clause 2 and Bibliography: update of normative and bibliographic references;
- subclause 3.4: update of abbreviations;

- subclause 4.1.2.1: update list of service request/response PDUs (Time Sync and Parameter ASEs/objects);
- subclause 4.1.8.6: update of Time Sync ASE/object;
- new subclause 4.1.8.7: new Parameter ASE/object;
- subclause 4.1.9.4: minor update of a common parameter;
- subclause 4.1.10.2: update/add object and services codes for Time Sync and Parameter ASEs;
- subclause 4.1.11: minor updates of error codes;
- new subclause 11.7: new QoS specification.

This bilingual version published in 2012-07 corresponds to the English version published in 2010-08.

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

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

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

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

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

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

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

NOTE 2 The revision of this standard will be synchronized with the other parts of the IEC 61158 series.

INTRODUCTION

This part of IEC 61158 is one of a series produced to facilitate the interconnection of automation system components. It is related to other standards in the set as defined by the “three-layer” fieldbus reference model described in IEC/TR 61158-1:2010.

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

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

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

INDUSTRIAL COMMUNICATION NETWORKS – FIELDBUS SPECIFICATIONS –

Part 6-2: Application layer protocol specification – Type 2 elements

1 Scope

1.1 General

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

This standard provides common elements for basic time-critical and non-time-critical messaging communications between application programs in an automation environment and material specific to Type 2 fieldbus. The term “time-critical” is used to represent the presence of a time-window, within which one or more specified actions are required to be completed with some defined level of certainty. Failure to complete specified actions within the time window risks failure of the applications requesting the actions, with attendant risk to equipment, plant and possibly human life.

This standard specifies interactions between remote applications and defines the externally visible behavior provided by the Type 2 fieldbus application layer in terms of

- a) the formal abstract syntax defining the application layer protocol data units conveyed between communicating application entities;
- b) the transfer syntax defining encoding rules that are applied to the application layer protocol data units;
- c) the application context state machine defining the application service behavior visible between communicating application entities;
- d) the application relationship state machines defining the communication behavior visible between communicating application entities.

The purpose of this standard is to define the protocol provided to

- a) define the wire-representation of the service primitives defined in IEC 61158-5-2:2010, and
- b) define the externally visible behavior associated with their transfer.

This standard specifies the protocol of the Type 2 fieldbus application layer, in conformance with the OSI Basic Reference Model (ISO/IEC 7498-1) and the OSI application layer structure (ISO/IEC 9545).

1.2 Specifications

The principal objective of this standard is to specify the syntax and behavior of the application layer protocol that conveys the application layer services defined in IEC 61158-5-2:2010.

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

1.3 Conformance

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

2 Normative references

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

IEC 61158-3-2:2007, *Industrial communication networks – Fieldbus specifications – Part 3-2: Data-link layer service definition – Type 2 elements*

IEC 61158-4-2:2010¹, *Industrial communication networks – Fieldbus specifications – Part 4-2: Data-link layer protocol specification – Type 2 elements*

IEC 61158-5-2:2010¹, *Industrial communication networks – Fieldbus specifications – Part 5-2: Application layer service definition – Type 2 elements*

IEC 61588:2009, *Precision clock synchronization protocol for networked measurement and control systems*

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

IEC 62026-3:2008, *Low-voltage switchgear and controlgear – Controller-device interfaces (CDIs) – Part 3: DeviceNet*

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

ISO/IEC 8802-3, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*

ISO/IEC 8824 (all parts), *Information technology – Abstract Syntax Notation One (ASN.1)*

ISO/IEC 8825-1, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

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

ISO/IEC 10646, *Information technology – Universal Multiple-Octet Coded Character Set (UCS)*

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

¹ To be published.

ISO 11898:1993², *Road vehicles – Interchange of digital information – Controller area network (CAN) for high-speed communication*

IEEE 802.1D, *IEEE standard for local and metropolitan area networks – Media Access Control (MAC) bridges*, available at <http://www.ieee.org>

IEEE 802.1Q, *IEEE standard for local and metropolitan area networks – Virtual bridged local area networks*, available at <http://www.ieee.org>

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

IETF RFC 1035, *Domain Names – Implementation and Specification*, available at <http://www.ietf.org>

IETF RFC 1122, *Requirements for Internet Hosts – Communication Layers*, available at <http://www.ietf.org>

IETF RFC 1759, *Printer MIB*, available at <http://www.ietf.org>

IETF RFC 2475, *An Architecture for Differentiated Services*, available at <http://www.ietf.org>

3 Terms, definitions, symbols, abbreviations and conventions

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

3.1 Terms and definitions from other ISO/IEC standards

3.1.1 Terms and definitions from ISO/IEC 7498-1

- a) abstract syntax
- b) application entity
- c) application process
- d) application protocol data unit
- e) application service element
- f) application entity invocation
- g) application process invocation
- h) application transaction
- i) presentation context
- j) real open system
- k) transfer syntax

3.1.2 Terms and definitions from ISO/IEC 9545

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

3.1.3 Terms and definitions from ISO/IEC 8824

- a) object identifier
- b) type

² A newer edition of this standard has been published, but only the cited edition applies.

- c) value
- d) simple type
- e) structured type
- f) component type
- g) tag
- h) Boolean type
- i) true
- j) false
- k) integer type
- l) bitstring type
- m) octetstring type
- n) null type
- o) sequence type
- p) sequence of type
- q) choice type
- r) tagged type
- s) any type
- t) module
- u) production

3.1.4 Terms and definitions from ISO/IEC 8825-1

- a) encoding (of a data value)
- b) data value
- c) identifier octets (the singular form is used in this standard)
- d) length octet(s) (both singular and plural forms are used in this standard)
- e) contents octets

3.2 Terms and definitions from IEC 61158-5-2

- a) application relationship
- b) client
- c) peer
- d) server

3.3 Additional terms and definitions

3.3.1

allocate

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

3.3.2

application

function or data structure for which data is consumed or produced

3.3.3

application objects

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

3.3.4

attribute

description of an externally visible characteristic or feature of an object

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

3.3.5

behavior

indication of how an object responds to particular events

3.3.6**boundary clock**

clock with more than a single PTP port, with each PTP port providing access to a separate PTP communication path

NOTE Boundary clocks are used to eliminate fluctuations produced by routers and similar network elements.

3.3.7**called**

service user or a service provider that receives an indication primitive or a request APDU

3.3.8**calling**

service user or a service provider that initiates a request primitive or a request APDU

3.3.9**class**

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

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

3.3.10**class attributes**

attribute that is shared by all objects within the same class

3.3.11**class code**

unique identifier assigned to each object class

3.3.12**class specific service**

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

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

3.3.13**client**

- a) object which uses the services of another (server) object to perform a task
- b) initiator of a message to which a server reacts

3.3.14**clock**

device providing a measurement of the passage of time since a defined epoch

NOTE There are two types of clocks in IEC 61588:2009, boundary clocks and ordinary clocks.

3.3.15**communication objects**

components that manage and provide a run time exchange of messages across the network

EXAMPLES Connection Manager object, Unconnected Message Manager (UCMM) object, Message Router object.

3.3.16**connection**

logical binding between application objects that may be within the same or different devices

NOTE Connections may be either point-to-point or multipoint.

3.3.17

connection ID

CID

identifier assigned to a transmission that is associated with a particular connection between producers and consumers, providing a name for a specific piece of application information

3.3.18

connection path

an octet stream which defines the application object to which a connection instance applies

3.3.19

connection point

buffer which is represented as a subinstance of an Assembly object

3.3.20

consume

act of receiving data from a producer

3.3.21

consumer

node or sink that is receiving data from a producer

3.3.22

consuming application

application that consumes data

3.3.23

cyclic

repetitive in a regular manner

3.3.24

device

physical hardware connected to the link

NOTE A device may contain more than one node.

3.3.25

device profile

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

3.3.26

end node

producing or consuming node

3.3.27

end point

one of the communicating entities involved in a connection

3.3.28

epoch

reference time defining the origin of a time scale

[IEC 61588:2009]

3.3.29

error

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

3.3.30**frame**

denigrated synonym for DLPDU

3.3.31**grandmaster clock**

clock which serve as the primary source of time to which all others (within a collection of IEC 61588 clocks) are ultimately synchronized.

3.3.32**instance**

the actual physical occurrence of an object within a class, identifying one of many objects within the same object class.

EXAMPLE California is an instance of the object class state.

NOTE The terms object, instance, and object instance are used to refer to a specific instance.

3.3.33**instance attribute**

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

3.3.34**instantiated**

object that has been created in a device

3.3.35**interoperability**

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

3.3.36**Keeper**

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

3.3.37**little endian**

Describes a model of memory organization which stores the least significant octet at the lowest address, or for transfer, which transfers the lowest order octet first

3.3.38**Lpacket**

a piece of application information that contains a size, control octet, tag, and link data.

NOTE Peer data-link layers use Lpackets to send and receive service data units from higher layers in the OSI stack.

3.3.39**management information**

network accessible information that supports the management of the Fieldbus environment

3.3.40**master clock**

single clock which serves as the primary source of time within each region, and which will in turn synchronize to other master clocks and ultimately to the grandmaster clock

NOTE A system of IEC 61588 clocks may be segmented into regions separated by boundary clocks.

3.3.41**member**

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

3.3.42

Message Router

object within a node that distributes messaging requests to appropriate application objects

3.3.43

multipoint connection

connection from one node to many

NOTE Multipoint connections allow messages from a single producer to be received by many consumer nodes.

3.3.44

network

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

3.3.45

object

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

3.3.46

object specific service

service unique to the object class which defines it

3.3.47

ordinary clock

IEC 61588 clock with a single PTP port

3.3.48

originator

client responsible for establishing a connection path to the target

3.3.49

point-to-point connection

connection that exists between exactly two application objects

3.3.50

Precision Time Protocol

PTP

name used for the time synchronization protocol.

3.3.51

produce

act of sending data to be received by a consumer

3.3.52

producer

node that is responsible for sending data

3.3.53

PTP message

IEC 61588 time synchronization message

NOTE There are five designated messages types defined by IEC 61588:2009: Sync, Delay_Req, Follow-up, Delay_Resp, and Management.

3.3.54

PTP port

logical access point for IEC 61588 communications to the clock containing the port

3.3.55**receiving**

service user that receives a confirmed primitive or an unconfirmed primitive, or a service provider that receives a confirmed APDU or an unconfirmed APDU

3.3.56**resource**

resource is a processing or information capability of a subsystem

3.3.57**sending**

service user that sends a confirmed primitive or an unconfirmed primitive, or a service provider that sends a confirmed APDU or an unconfirmed APDU

3.3.58**server**

- a) role of an AREP in which it returns a confirmed service response APDU to the client that initiated the request
- b) object which provides services to another (client) object

3.3.59**service**

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

3.3.60**synchronized clocks**

(to a specified uncertainty) two clocks which have the same epoch and for which measurements of any time interval by both clocks differ by no more than the specified uncertainty

NOTE The timestamps generated by two synchronized clocks for the same event will differ by no more than the specified uncertainty.

3.3.61**System Time**

absolute time value as defined by CPF2 time synchronization in the context of a distributed time system where all devices have a local clock that is synchronized with a common master clock

NOTE In the context of CPF2, System Time is a 64-bit integer value in units of nanoseconds with a value of 0 corresponding to the date 1970-01-01.

3.3.62**target**

end-node to which a connection is established

3.3.63**temporary node**

transient node

3.3.64**transaction id**

field within a UCMM header that matches a response with the associated request

3.3.65**Unconnected Message Manager****UCMM**

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

3.3.66

unconnected service

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

3.3.67

vendor ID

identification of each product manufacturer/vendor by a unique number

NOTE Vendor IDs are assigned by ODVA, Inc.

3.4 Abbreviations and symbols

ASCII	American Standard Code for Information Interchange	
CID	connection ID	
DLL	data-link layer	
DSCP	DiffServ Codepoint	
IGMP	Internet Group Management Protocol (see IETF RFC 1112, IETF RFC 2236)	
IP	Internet Protocol (see IETF RFC 791)	
IPv4	Internet Protocol version 4 (see IETF RFC 791)	
IPv6	Internet Protocol version 6 (see IETF RFC 791)	
OSI	open systems interconnection (see ISO/IEC 7498-1)	
PDU	protocol data unit	
PHB	per-hop behavior	
PTP	Precision Time Protocol	[IEC 61588:2009]
QoS	quality of service	
Rcv	receive	
Rx	receive	
SDU	service data unit	
SEM	state event matrix	
STD	state transition diagram, used to describe object behavior	
TCP	Transmission Control Protocol (see IETF RFC 793)	
ToS	type of service	
TPDU	transport protocol data unit	
Tx	transmit	
UDP	User Datagram Protocol (see IETF RFC 768)	
Xmit	transmit	
CM_API	actual packet interval	
O2T or O⇒T	originator to target (connection parameters)	
CM_RPI	requested packet interval	
TUI	table unique identifier	
T2O or T⇒O	target to originator (connection parameters)	

3.5 Conventions

3.5.1 General concept

The FAL is defined as a set of object-oriented ASEs. Each ASE is specified in a separate subclause. Each ASE specification is composed of three parts: its class definitions, its services, and its protocol specification. The first two are contained in IEC 61158-5-2:2010. The protocol specification for each of the ASEs is defined in this standard.

The class definitions define the attributes of the classes supported by each ASE. The attributes are accessible from instances of the class using the Management ASE services specified in IEC 61158-5-2:2010. The service specification defines the services that are provided by the ASE.

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

3.5.1.1 Attribute specification

Attributes are defined in an Attribute Table using the format and terms defined in Figure 1.

Attribute ID	Name	Data type	Semantics of values

Figure 1 – Attribute table format and terms

The **Attribute ID** shall be a unique integer identification value assigned to an attribute. The valid ranges for Attribute IDs shall be as specified in 4.1.10.1.3. The Attribute ID shall identify the particular attribute being accessed.

Name shall refer to the name assigned to the attribute. An attribute may contain sub-elements, which may also have names, as is the case with the **STRUCT** of data type. The attribute name shall be the name which appears first, or at the top row in the name column (the row that contains the Attribute ID).

Every attribute shall be assigned a **Data type** which shall be either an elementary or a derived data type. The data types specified for the defined attributes shall be used in all implementations.

NOTE The elementary data types are defined in IEC 61158-5-2:2010.

Semantics of values shall specify the meaning of the value(s) of the attribute. If this information needs more room than can fit in the table it will immediately follow the Class Attribute table. Included in the Class Attribute table will be an appropriate reference to this information.

If a Class Attribute is *optional*, then a default value or a special case processing method shall be defined such that the Client (Requester) can process the error message that occurs when accessing those objects that choose not to implement the class attribute.

3.5.1.2 Common services

3.5.1.2.1 Service_PDU definitions

Each service has unique parameters for request and response. The service request and response parameters shall be defined using service Request/Response parameter tables as defined in Figure 2.

Name	Type	Semantics of values

Figure 2 – Service request/response parameter

Name shall refer to the name given to the service request/response parameter.

Type shall specify the data type of the service request/response parameter.

Semantics of values shall specify the meaning of the values of the service request/response parameter, e.g. "the value is counts of microseconds."

3.5.1.2.2 Get_Attribute_All response

3.5.1.2.2.1 General definition

When the Get_Attribute_All common service is included in the list of supported System/Object Management services for an object class, then the **Get_Attribute_All** response shall be detailed for this class : the sequence or order of the data returned in the Service_ResponsePDU shall be specified. There are three ways in which the Get_Attribute_All Service_ResponsePDU may be specified:

- list the ordering of the attributes in the response message;
- specify the actual data array of the response message;
- combine the above two methods such that the actual data array also contains the attribute numbers.

Whichever method is used, the rules specified in Table 1 shall be adhered to when specifying the Get_Attribute_All Service_ResponsePDU of an object class for both the Class Attributes and the Instance Attributes.

Table 1 – Get_Attribute_All response service rules

Rule number	Rule
1	If the definition of the Get_Attribute_All response includes optional attributes, then default values shall be specified in the response description. These defaults shall be used if an implementation does not support the optional attributes. If any of the following optional attributes are included in an object specification, but not supported in the implementation of the object, then the following shall be adhered to: – if the class attribute “Optional attribute list” is not supported, the default value of zero shall be inserted into the response array and no optional attribute numbers shall follow; – if the class attribute “optional service list” is not supported, the default value of zero shall be inserted into the response array and no optional service numbers shall follow.
2	If new attributes are added to an existing object, those attributes shall be added to the end of the response attribute list or data array to ensure compatibility with different object revisions.
3	Whichever method is used to specify the response, it shall be done in such a way as to be unambiguous, including rules to deal with variable length fields and padding.
4	The Get_Attribute_All response shall include only the open attributes; it shall not include any vendor specific attributes.

Table 2 is an example of the attribute ordering method of specifying the service data portion of a Get_Attribute_All response for class level attributes of an object which supports **optional gettable** class attributes 1,2,3 and 4.

Table 2 – Example class level object/service specific response data of Get_Attribute_All

Class attribute ID	Attribute name and default value
1	Revision, default = 0x0001
2	Max Instance, default = 0x0000
3	Number of Instances, default = 0x0000
4	Attribute list, number of attributes, default = 0x0000

Table 3 is an example of the data array method of specifying the service data portion of a Get_Attribute_All response for class level attributes of an object which supports **optional gettable** class attributes 1, 2, 3 and 4.

Table 3 – Example Get_Attribute_All data array method

Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Revision (low octet) Default = 1							
1	Revision (high octet) Default = 0							
2	Max Instance (low octet) Default = 0							
3	Max Instance (high octet) Default = 0							
4	Number of Instances (low octet) Default = 0							
5	Number of Instances (high octet) Default = 0							
6	Optional Attribute List : number of attributes (low octet) Default = 0							
7	Optional Attribute List : number of attributes (high octet) Default = 0							
8	Optional Attribute List : optional attribute #1 (low octet)							
9	Optional Attribute List : optional attribute #1 (high octet)							
2n + 6	Optional Attribute List : optional attribute #n (low octet)							
2n + 7	Optional Attribute List : optional attribute #n (high octet)							

3.5.1.2.2.2 Revisions

The defined Get_Attribute_All response for an object may increase in size with each revision of the object; however, to insure interoperability, the format of the first part of the response shall remain parseable by a client of the older revision of the object. Clients (Requesters) need not make use of this compatibility requirement.

NOTE The Revision class attribute is not the revision of an implementation (which is reflected in the Identity object, Minor/Major revision status bits), but the revision of the *class definition*.

3.5.1.2.3 Set_Attribute_All request

3.5.1.2.3.1 General definition

When the Set_Attribute_All common service is included in the list of supported common services, then the format of the **Set_Attribute_All request** shall be included in the object specification. The sequence or order of the data supplied in the Service Data portion of the request shall be specified. There are three ways in which the Set_Attribute_All request may be specified:

- list the order of the attributes in the request message;
- specify the actual data array of the request message;
- combine the above two methods such that the actual data array also contains the attribute numbers.

Whichever method is used, the rules specified in Table 4 shall be adhered to when specifying an object's Set_Attribute_All request in the object specification for both the Class Attributes and the Instance Attributes.

Table 4 – Set_Attribute_All request service rules

Rule number	Rule
1	An object shall support the Set_Attribute_All service only if all settable attributes shown in the Set_Attribute_All request are implemented as settable.
2	If new settable attributes are added to an existing object, those attributes shall be added to the end of the request attribute list or data array.
3	Whichever method is used to specify the response, it shall be done in such a way as to be unambiguous, including rules to deal with variable length fields and padding.
4	The Set_Attribute_All request shall include only the open attributes. It shall not include any vendor specific attributes.

Table 5 is an example of the attribute ordering method of specifying the service data portion of a Set_Attribute_All request for instance level attributes of an object which supports required settable instance attributes 7,8,9,10,11 and 12.

Table 5 – Example Set_Attribute_All attribute ordering method

Class Attribute ID	Attribute name
7	Output Range
8	Value Data Type
9	Fault State
10	Idle State
11	Fault Value
12	Idle Value

Table 6 is an example of the data array method of specifying the service data portion of a Set_Attribute_All request for instance level attributes of an object which supports required settable instance attributes 7, 8, 9, 10, 11 and 12.

Table 6 – Example Set_Attribute_All data array method

Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Output Range							
1	Value Data Type							
2	Fault State							
3	Idle State							
4	Fault Value (low octet)							
5	Fault Value (high octet)							
6	Idle Value (low octet)							
7	Idle Value (high octet)							

3.5.1.2.3.2 Revisions

A server processing a Set_Attribute_All request may need to process more data than is expected by the server if the client has implemented a newer revision of this object while the server an older revision. As a result, the server object may have to process a Set_Attribute_All request that contains more data because the additional attributes were not recognized. If the server receives more data in a Set_Attribute_All request than it expects the server shall respond with a general status code equal to *Too much data* (0x15).

NOTE The Revision class attribute is not the revision of an implementation (which is reflected in the Identity object, Minor/Major revision status bits), but the revision of the *class definition*.

3.6 Conventions used in state machines

3.6.1 State machine conventions for Type 2

3.6.1.1 General

State changes may be triggered by events (internal or external) or service invocations. Reaction to service invocations may depend on the value(s) of the attribute(s) accessed by the service.

Behaviour of the state machine shall be defined for combinations of :

- the *event* the machine receives;
- the *state* the machine is in when it receives notification of a state-changing event.

To define behaviour in these terms, a State Transition Diagram (STD) and a State Event Matrix (SEM) are used when applicable in the state machine specification.

3.6.1.2 State Transition Diagram STD

An **event** is an external stimulus that may cause a state transition. An STD graphically illustrates the states of an object and includes events, service calls and changes of attributes that cause it to transition to another state. Figure 3 shows an example of an STD.

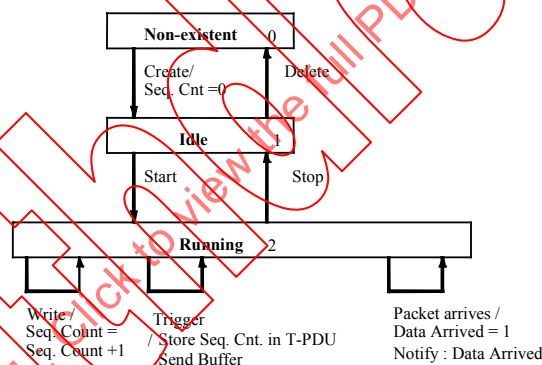


Figure 3 – Example of an STD

NOTE Event that is the primary cause of the State Transition is followed by “/” mark. Notification to upper layer is marked by “Notify”.

3.6.1.3 State Event Matrix

A **state** is the current active mode of operation of the state machine object (e.g. Running, Idle).

A state event matrix is a table that lists all possible events, services and changes in attribute values that initiate a state change, and indicates the response by the object to the event based on the state of the object when it receives notification of that event.

State event matrix format is as shown in Table 7.

Table 7 – State event matrix format

Event	State		
	State 1		State n
Event A description	Function triggered by event A in State 1 (if any) Notification to FAL user (if any) Transition to another state (if any)		Function triggered by event A in State n (if any) Notification to FAL user (if any) Transition to another state (if any)
.....			
Event X description	Function triggered by event X in State 1 (if any) Notification to FAL user (if any) Transition to another state (if any)		Function triggered by event X in State n (if any) Notification to FAL user (if any) Transition to another state (if any)
In absence of function, notification or transition, the corresponding entry shall not be made in the table.			

3.6.1.4 Example state event matrix

Table 8 shows an example of a state machine with three states

- *Non-existent*: the object has not yet been created, objects transition to the existent state via the create service (if the object may be dynamically created) or at power-up (if the object is fixed by design/implementation);
- *Idle*: the object accepts services (e.g. Get_Attribute_Single), but does not produce or consume data onto or from the link;
- *Running*: the object is performing all its specified functions.

Table 8 – Example state event matrix

Event	State		
	Non-existent	Idle	Running
Create	Transition to Idle	Error: Object already exists.	Error: Object already exists.
Delete	Error: Object does not exist. (General Error Code 0x16)	Transition to Non-existent	Error: Object State Conflict (General Error Code 0x0C)
Start	Error: Object does not exist. (General Error Code 0x16)	Transition to Running	Error: Object State Conflict (General Error Code 0x0C)
Stop	Error: Object does not exist. (General Error Code 0x16)	Error: Object State Conflict (General Error Code 0x0C)	Transition to Idle
Get_Attribute_Single	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Validate/service the request. Return response
Set_Attribute_Single	Error: Object does not exist. (General Error Code 0x16)	Validate/service the request. Return response	Error: Object State Conflict (General Error Code 0x0C)

4 Abstract syntax

4.1 FAL PDU abstract syntax

4.1.1 General

FAL_PDU shall be either a UCMM_PDU or a Transport_PDU.

UCMM_PDU is used to convey information for connection-less services.

Transport_PDU is used to convey information for connection-oriented services.

A connected message assumes previously negotiated resources and parameters at its source, its destination(s), and any intermediate transit points. These resources are referenced by a unique connection identifier and do not need to be contained in each message, only the connection ID is needed to identify the message and refer to its related parameters, thus giving significant savings in message efficiency.

An unconnected message provides a means to communicate on the local link without previously negotiated resources at the destination so it shall carry full destination ID details, internal data descriptors and full source ID details if a reply is requested. Unconnected messages are used mainly to create connections.

The unconnected service is provided by the Unconnected Message Manager (UCMM). Messages received through the UCMM are forwarded to the Message Router (MR), which direct them to the appropriate internal object for execution. Connections are established by specific unconnected messages sent through the Connection Manager (CM) using UCMM services. Connections may be established either to the Message Router (for messaging purpose), or directly to an application object. Connection target is specified using a connection path, and may be either on the local or a remote link, through several intermediate router nodes. Once a connection is established with an application object, the UCMM, MR and CM are no longer required, since data will be exchanged directly with the connected object, based on the corresponding connection ID. Connected messages sent to the Message Router will be forwarded to the appropriate internal object for execution.

4.1.2 PDU structure

4.1.2.1 UCMM_PDU structure

UCMM_PDU is a sequence of a UCMM_Header, followed by either OM_Service (Object Management ASE) or CM_Service (Connection Manager ASE).

OM_Service shall be either OM_Request or OM_Response.

OM_Request shall consist of MR_Request_Header and a Service_RequestPDU.

Service_RequestPDU shall be one of the following :

- Get_Attribute_Single_RequestPDU
- Get_Attribute_All_RequestPDU
- Get_Attribute_List_RequestPDU
- Set_Attribute_Single_RequestPDU
- Set_Attribute_All_RequestPDU
- Set_Attribute_List_RequestPDU
- Reset_RequestPDU
- Create_RequestPDU
- Delete_RequestPDU
- Start_RequestPDU
- Stop_RequestPDU
- Find_Next_Object_Instance_RequestPDU
- NOP_RequestPDU
- Apply_Attributes_RequestPDU
- Save_RequestPDU

- Restore_RequestPDU
- Group_Sync_RequestPDU
- Add_AckData_Path_RequestPDU
- Remove_AckData_Path_RequestPDU
- Get_Enum_String_RequestPDU
- Connection_Bind_RequestPDU
- Producing_Application_Lookup_RequestPDU

OM_Response shall consist of MR_Response_Header and a Service_ResponsePDU.

Service_ResponsePDU shall be one of the following :

- Get_Attribute_Single_ResponsePDU
- Get_Attribute_All_ResponsePDU
- Get_Attribute_List_ResponsePDU
- Set_Attribute_Single_ResponsePDU
- Set_Attribute_All_ResponsePDU
- Set_Attribute_List_ResponsePDU
- Reset_ResponsePDU
- Create_ResponsePDU
- Delete_ResponsePDU
- Start_ResponsePDU
- Stop_ResponsePDU
- Find_Next_Object_Instance_ResponsePDU
- NOP_ResponsePDU
- Apply_Attributes_ResponsePDU
- Save_ResponsePDU
- Restore_ResponsePDU
- Group_Sync_ResponsePDU
- Add_AckData_Path_ResponsePDU
- Remove_AckData_Path_ResponsePDU
- Get_Enum_String_ResponsePDU
- Connection_Bind_ResponsePDU
- Producing_Application_Lookup_ResponsePDU

CM_Service shall be either CM_Request or CM_Response.

CM_Request shall consist of MR_Request_Header and a CM_RequestPDU.

CM_RequestPDU shall be one of the following :

- Forward_Open_RequestPDU
- Forward_Close_RequestPDU
- Large_Forward_Open_RequestPDU
- Unconnected_Send_RequestPDU
- Get_Connection_Data_RequestPDU

- Search_Connection_Data_RequestPDU
- Get_Object_Owner_RequestPDU

CM_Response consists of MR_Response_Header and a CM_ResponsePDU.

CM_ResponsePDU shall be one of the following :

- Forward_Open_ResponsePDU
- Forward_Close_ResponsePDU
- Large_Forward_Open_ResponsePDU
- Unconnected_Send_ResponsePDU
- Get_Connection_Data_ResponsePDU
- Search_Connection_Data_ResponsePDU
- Get_Object_Owner_ResponsePDU

4.1.2.2 Transport_PDU structure

Transport_PDU is a sequence of a Transport_Header, followed by either OM_Service (Object Management ASE) or Application Data.

OM_Service is defined in 4.1.2.1 above.

Application Data comes from the source to which the connection has been made.

4.1.3 UCMM_PDUs

All UCMM_PDUs shall be sent and received using fixed tag 0x83 or Management tag 0x88. When a device becomes powered, the UCMM shall invoke the `DLL_enable_fixed_request` service of the data-link layer to request that fixed tag 0x83 or 0x88 Lpackets be routed to the UCMM. All sending and receiving of TPDUs shall use the `DLL_fixed_request.req` and `DLL_fixed_request.ind` service primitives of the data-link layer, respectively. The packet parameter of these services shall be prefixed with the following header to form the Transport PDU before sending to the data-link layer. The format of the UCMM_PDU Header is shown in Table 9.

Table 9 – UCMM_PDU header format

Parameter name	Format
command_code	USINT
Timeout	USINT
TransactionID	UINT

The `command_code` shall specify the type of UCMM_PDU as shown in Table 10. UCMM packets received with a reserved `command_code` shall be discarded without acknowledgement.

Table 10 – UCMM command codes

Command_code	Description
0	Reserved
1	acknowledge a request
2	request with retry until acknowledged
3	response with retry until acknowledged

Command_code	Description
4	request with no acknowledge and no response
5	acknowledge a response
6	response which shall not retry (no acknowledge)
7	request with retry until response (no acknowledge)
8	Request which shall not retry and shall cause a code 6 response
9 – 255	Reserved

The timeout field shall specify the duration of the transaction in an 8-bit floating-point format. The most significant 5 bits of the field shall be an unsigned exponent that is not biased. The least significant 3 bits shall be the least significant 3 bits of a 4 bit unsigned mantissa. The most significant bit of the mantissa shall be 1 and shall not appear explicitly in the 8-bit representation. The binary point shall be positioned between the implied 1 and the rest of the mantissa. The units of the transaction duration computed using the timeout field shall be in milliseconds.

NOTE Using ANSI C precedence rules, the number of 0,125 ms ticks is $(8 \mid \text{timeout} \& 7) \ll (\text{timeout} \gg 3 \& 31)$.

The transactionID field shall be composed of two sub-fields:

- the least significant 10 bits, RECORD, shall identify a specific transaction;
- the most significant 6 bits, SEQUENCE, shall be used for duplicate detection. It shall be incremented for every unique message on this RECORD; however it shall not be incremented on a retry.

Only one message shall be outstanding for a given RECORD.

If multiple outstanding messages are required then multiple RECORDs are required.

When an I'm alive fixed tag packet is received from a node (see IEC 61158-4-2:2010, Station Management), all UCM transactions with that node shall be aborted.

4.1.4 Transport_Headers

4.1.4.1 General

Contents of Transport headers varies depending on the class of transport selected during the connection establishment.

4.1.4.2 Class 0 (null or base)

The class 0 transport header shall be an unsigned 16-bit number. This header shall be written to the TPDU buffer by the transport and shall be simply discarded by the transport from the packet coming from the consumer. The format of the Header is shown in Table 11.

Table 11 – Transport class 0 header

Parameter name	Format
don't_care	UINT

4.1.4.3 Class 1 (duplicate detection)

The class 1 transport header shall be an unsigned 16-bit sequence count. This header shall be written to the TPDU buffer by the transport and shall be read by the transport from the packet coming from the consumer. The format of the Header is shown in Table 12.

Table 12 – Transport class 1 header

Parameter name	Format
sequence_count	UINT

4.1.4.4 Class 2 (acknowledged)

Like the class 1 transport header, the class 2 transport header shall be a 16-bit sequence count. This header shall be written to the TPDU buffer by the transport and shall be read by the transport from the packet coming from the consumer. The format of the Header is shown in Table 13.

Table 13 – Transport class 2 header

Parameter name	Format
sequence_count	UINT

4.1.4.5 Class 3 (verified)

Like the class 1 transport header, the class 3 transport header shall be a 16-bit sequence count. This header shall be written to the TPDU buffer by the transport and shall be read by the transport from the packet coming from the consumer. The format of the Header is shown in Table 14.

Table 14 – Transport class 3 header

Parameter name	Format
sequence_count	UINT

4.1.4.6 Class 4 (non-blocking), class 5 (non-blocking, fragmenting) and class 6 (multipoint connection, fragmenting)

These classes of transports all use the header format shown in Table 15. Classes 4 and 5 have two headers in each TPDU, one Sender and one Receiver. Class 6 uses a Sender header in the Client to Server direction and a Receiver header in the Server to Client direction.

Table 15 – Classes 4 to 6 header format

Command			Sequence number	Data	
MSB	Bits 15 – 12		Bits 11 – 8	Bits 7 – 0	LSB
Value:	Meaning in Sender Header:	Meaning in Receiver Header:	Increments from 0 to 15, then rolls over to zero again. Used in Sender header to distinguish a retry from a new transmission. Used in Receiver header to indicate which sender TPDU is being ACK'd or NAK'd.	For "First": Number of 256 octet blocks (minus 1) to allocate to hold complete message. For "NAK": Reason code. For others: Set to zero when generating; ignore when receiving.	
0.	Null	ACK			
1.	Only	NAK			
2.	First	Not Initialized			
3.	Middle	Undefined			
4.	Last	Undefined			

Command		Sequence number	Data	
MSB	Bits 15 – 12	Bits 11 – 8	Bits 7 – 0	LSB
5. to 15 are undefined.				

The Command identifies the use of the TPDU by the transport. The Command field is defined differently for the Sender and the Receiver header.

Sender header commands :

Null = no data in this Transport PDU

Only = this is the Transport PDU conveying all of the Application data

First / Middle / Last = position of this Transport PDU within the segmented Application data

The Sequence Number is used in the Sender header to distinguish a retry from a new transmission. It is used in Receiver header to indicate which sender TPDU is being ACK'd or NAK'd. This is **NOT** used for applications request/response matching. This is only used by the transport state machines.

The Data is used by one Sender command and one Receiver command. It is used in the *First* Sender command to indicate the number of 256 octet blocks (minus 1) the Receiver shall allocate for the complete application message. That is, application messages of size 0 to 256 octets shall have the *First* Data field set to 0; 257 to 512, to 1; and so on up to 65 536 octets. It is used in the NAK Receiver command to provide a reason code:

Class 4 (the Point-to-Point non-Blocking transport) uses two headers in each TPDU, followed by the application data. The Sender Header is first in the TPDU, the Receiver Header is second, followed by the application data. Class 4 only uses the *Null* and *Only* Sender Commands (numbered 0 and 1).

Class 5 (the Point-to-Point Fragmenting non-Blocking transport) uses two headers in each TPDU, followed by the application data. The Sender Header is first in the TPDU, the Receiver Header is second, followed by the application data. Class 5 uses all of the Sender Commands.

Class 6 (the Multipoint connection Fragmenting transport) uses one header in each TPDU, followed by the application data. Class 6 uses all of the Sender Commands. The Sender Header is used in the Client-to-Server TPDU and the Receiver Header is used in the Server to Client TPDU.

4.1.4.7 Exclusive owner O⇒T data format

Exclusive owner connections shall have either of two real-time transfer formats:

- 32-bit header, fixed size;
- no header, variable size.

The 32-bit header prefixed to the real-time data shall be in the form shown in Table 16.

Table 16 – Real-time data header – exclusive owner

Parameter	Format	Size (bits)
Run_idle	RUN_IDLE	1
Reserved	UDINT	31

The `run_idle` flag (bit 0) shall be set (1 = RUN) to indicate that the following data shall be sent to the target application. It shall be clear (0 = IDLE) to indicate that the idle event shall be sent to the target application. The `reserved` field (bits 1-31) shall be reserved and set to 0.

If the `no_header` transfer format is used, the reception of a packet with data beyond the transport header shall indicate the RUN mode. If the packet is truncated after the transport header, the target shall be sent an idle event.

4.1.4.8 Redundant owner O⇒T data format

4.1.4.8.1 General format

Redundant owner shall have a 32-bit header prefixed to the real-time data. This header shall be in the form shown in Table 17.

Table 17 – Real-time data header– redundant owner

Parameter	Format	Size (bits)
Run_idle	RUN_IDLE	1
COO	BOOLEAN	1
ROO	USINT	2
Reserved	UDINT	28

4.1.4.8.2 Run_idle flag

The `run_idle` flag (bit 0) shall have the same meaning as it does in an exclusive owner connection and shall be one of RUN or IDLE. The reserved field (bits 4-31) shall be reserved and set to 0.

4.1.4.8.3 Claim output ownership (COO) flag

The COO flag shall be set (1) when an originator application wants its connection to be the owning connection of the target application. The COO flag shall be reset (0) when an originator application does not want its connection to be the owning connection of the target application. When the owning connection resets (0) its COO flag, its sibling connections shall be checked for a set (1) COO flag. The new owner shall be any of the connections that have their COO flag set.

NOTE This results in undefined behaviour if more than one other connection has its COO flag set.

4.1.4.8.4 Ready for ownership of outputs (ROO) priority value

The ROO priority value shall be non-zero when an originator application does not want to force its connection to be the owning connection of the target application, but is ready to be the owning connection should there be no originator applications claiming to be the owning connection. The ROO priority value shall be zero when the originator application does not want to be the owning connection of the target connection and is not to be the owning connection should there be no originator application claiming to be the owning connection. The ROO priority value shall be used only when the COO flag is reset.

The value of the ROO field can range from 0 to 3. The originator applications shall each determine a unique non-zero ROO value.

4.1.4.8.5 Determining the owning connection

The originator applications shall determine among themselves which originator application has the owning connection. The owning connection shall be determined by the originator application that sets its COO flag. In situations where multiple originator applications have

their COO flag set or where no originator connections have their COO flag set, the following rules shall be applied by the target transport to determine the owning connection.

- There shall be no owning connection until an originator application sends a real-time packet with the COO flag set;
- If there is only one originator application which had the COO flag set in its last real-time packet, that originator application shall have the owning connection;
- If there are multiple originator applications which had the COO flag set in its last real-time packet, the last originator application that transitioned its COO flag from reset to set shall have the owning connection.
- If the originator application with the owning connection resets its COO flag, closes its connection, or if that connection times out, and no other originator applications have their COO flags set, the originator application with the highest non-zero ROO priority value shall have the owning connection.
- If all of the originator applications have their COO flags reset and ROO priority values set to zero, there shall be no owning connection.
- When the first real-time packet containing a set COO flag is received by the target transport, the originator application that sent the real-time packet shall have the owning connection.

4.1.4.8.6 Transporting events and data to a target application

The connection related events from each of the redundant owner connections shall be combined using the following rules such that the target application sees only a single exclusive owner connection:

- Until an owning connection is initially determined, the transport shall not indicate real-time data reception to the target application;
- If an owning connection is determined, the transport shall indicate the event consistent with the owning connections real-time data to the target application. If the Run/Idle flag is reset in the real-time data for the owning connection, the transport shall indicate the idle state to the target application. If the Run/Idle flag is set in the real-time data for the owning connection, the transport shall indicate the run state and the real-time data to the target application;
- If an owning connection had been previously determined, but no originator is currently claiming or ready for ownership, the transport shall indicate idle state to the target application;
- If all the redundant connections are closed or have been timed out, the target transport shall indicate the event consistent with the last connection to have been closed or timed out to the target application.

4.1.5 CM_PDUs

4.1.5.1 General

Connection establishment and maintenance services are provided by the Connection Manager. They are

CM_Forward_Open (corresponds to Forward_Open and Ex_Forward_Open PDUs)
 CM_Forward_Close (corresponds to Forward_Close PDUs)
 CM_Unconnected_Send (corresponds to Unconnected_Send PDUs)
 CM_Get_Connection_Data (corresponds to Get_Connection_Data PDUs)
 CM_Search_Connection_Data (corresponds to Search_Connection_Data PDUs)
 CM_Get_Object_Owner (corresponds to Get_Object_Owner PDUs)

4.1.5.2 Connection Manager

The Connection Manager object is used to manage the establishment and maintenance of communication connections.

The Connection Manager objects at different nodes shall communicate using the UCMM services described in IEC 61158-5-2:2010. The TPDU's with which peer Connection Managers communicate shall be derived from the services of the Connection Manager.

4.1.5.3 Forward_Open

4.1.5.3.1 General

The response to the Forward_Open_Request_PDU is a Forward_Open_Response_PDU. The Forward_Open response shall contain the original connection serial number and owner vendor and serial number and the connection IDs (CID) necessary to complete the connection if successful and error information if the connection failed.

NOTE 1 The Forward_Open request sets up network, transport, and application connections. An application connection consists of a single transport connection, and one or two network connections that are in turn comprised of multiple link connections. Each port segment in the connection path uses a link connection. The Forward_Open service between two devices builds one or two link connections as specified by the network connection parameter and the requested packet intervals (CM_RPI). Since up to two network connections can be required for a single transport connection, they are differentiated by the O⇒T and T⇒O designations; O⇒T means originator to target, and T⇒O means target to originator.

NOTE 2 The processing of a single Forward Open service may result in the creation of multiple active Connection object instances.

4.1.5.3.2 Format of Forward_Open request

The format of the Forward_Open request TPDU shall be of the form shown in Table 18.

Table 18 – Forward_Open request format

Parameter name	Format
priority_and_tick	SWORD
connection time-out ticks	USINT
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_connection_parameters	UINT
T2O_CM_RPI	UDINT
T2O_connection_parameters	UINT
xport_type_and_trigger	SWORD
connection_path_size	USINT
connection_path	Padded EPATH

4.1.5.3.3 Format of Forward_Open response if success

If the `status` parameter of the `CM_open_response` is zero (no error), the format of the Forward_Open response TPDU shall be of the form shown in Table 19.

Table 19 – Forward_Open_Good response format

Parameter name	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
O2T_CM_API	UDINT
T2O_CM_API	UDINT
Application_reply_size; (in 16-bit words)	USINT
Reserved	USINT
application_reply[]	UINT

Success shall be returned when the connection requested has been established from this point forward in the path. This response also shall indicate the connection serial number and the actual packet rate of the connection. Once the successful response has been received, the connection shall be open from this point forward in the path. Targets shall wait at least 10 seconds after sending the CM_Forward_Open_Good_Response for the first packet on a connection.

4.1.5.3.4 Format of Forward_Open response if failure

If the status parameter of the CM_open_response is not zero (error), the format of the Forward_Open response TPDU shall be of the form shown in Table 20.

Table 20 – Forward_Open_Bad response format

Parameter name	Format
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size; (in 16-bit words)	USINT
Reserved	USINT

This format shall be used for all failures of the connection system. The requested connection shall not be established, and the object specific status words shall contain information about the reason for the failure. The remaining_path_size shall contain the length of the path at the point the connection failed. This information can be used to debug the problem.

In the failure response, the remaining remaining_path_size shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the “pre-stripped” size or 0 for the remaining remaining_path_size.

A duplicate Forward_Open service shall be defined as a Forward_Open service whose vendor_ID, connection_serial_number, and originator_serial_number match an existing connection’s parameters. If the duplicate Forward_Open service is a null Forward_Open service (defined as the connection type in both the O2T and T2O network connection parameter fields are NULL), then the Forward_Open service shall be forwarded to the application for further processing. Null Forward_Open requests may be used to reconfigure the connection. The Connection Manager in the intermediate nodes need not allocate additional resources for a duplicate Forward_Open request since the resources have

already been allocated. If the duplicate Forward_Open request is not NULL, then an general status = 0x01, extended status = 0x0100 shall be returned.

The suggested originator behavior in the duplicate Forward_Open case should be to either close and re-establish the connection or wait for the connection to time-out and then establish the connection again. It shall be recognized that in the latter case, it shall take the connection 60 seconds (first data time-out) to time-out before the connection can be re-established.

4.1.5.4 Large_Forward_Open

4.1.5.4.1 General

The Large_Forward_Open shall have the same function as the Forward_Open except that it shall allow the establishment of connections larger than 511 octets.

The response to the Large_Forward_Open_Request_PDU is an Large_Forward_Open_ResponsePDU. The Large_Forward_Open response shall contain the original connection serial number and owner vendor and serial number and the connection IDs (CID) necessary to complete the connection if successful and error information if the connection failed.

NOTE The Large_Forward_Open request is very similar to the Forward_Open request.

4.1.5.4.2 Format of Large_Forward_Open request

The format of the Large_Forward_Open request TPDU shall be of the form shown in Table 21.

Table 21 – Large_Forward_Open request format

Parameter name	Format
priority_and_tick	SWORD
connection time-out ticks	USINT
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_ex_connection_size	UINT
O2T_connection_parameters	UDINT
T2O_CM_RPI	UDINT
T2O_ex_connection_size	UINT
T2O_connection_parameters	UDINT
xport_type_and_trigger	SWORD
connection_path_size	USINT
connection_path	Padded EPATH

The connection size field in the two connection_parameters shall be reserved in this case and set to zero. The ex_connection_size parameters shall be used instead to specify the maximum size of the connection (up to 65 535).

4.1.5.4.3 Format of Large_Forward_Open response if success

If the `status` parameter of the `CM_open_response` is zero (no error), the format of the `Large_Forward_Open` response TPDU shall be of the form shown in Table 22.

Table 22 – Large_Forward_Open_Good response format

Parameter name	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
O2T_CM_API	UDINT
T2O_CM_API	UDINT
Application_reply_size; (in 16-bit words)	USINT
Reserved	USINT
application_reply[]	UINT

Success shall be returned when the connection requested has been established from this point forward in the path. This response also shall indicate the connection serial number and the actual packet rate of the connection. Once the successful response has been received, the connection shall be open from this point forward in the path.

4.1.5.4.4 Format of Large_Forward_Open response if failure

If the `status` parameter of the `CM_open_response` is not zero (error), the format of the `Large_Forward_Open` response TPDU shall be of the form shown in Table 23.

Table 23 – Large_Forward_Open_Bad response format

Parameter name	Format
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size; (in 16-bit words)	USINT
Reserved	USINT

Either a target device or an intermediate router shall return an Invalid Connection Size extended error code, along with the maximum connection size supported additional status word, if the connection size requested is not supported while processing a `Large_Forward_Open` request.

This format shall be used for all failures of the connection system. The requested connection shall not be established, and the object specific status words shall contain information about the reason for the failure. The `remaining_path_size` shall contain the length of the path at the point the connection failed. This information can be used to debug the problem.

In the failure response, the remaining `remaining_path_size` shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the “pre-stripped” size or 0 for the remaining `remaining_path_size`.

The suggested originator behavior in the duplicate Forward_Open case should be to either close and re-establish the connection or wait for the connection to time-out and then establish the connection again. It shall be recognized that in the latter case, it shall take the connection 60 seconds (first data time-out) to time-out before the connection can be re-established.

4.1.5.5 Forward_Close

4.1.5.5.1 General

The Forward_Close request shall remove a connection from all the nodes participating in the original connection. The Forward_Close shall be sent between Connection Managers as specified in the connection_path. The Forward_Close request shall cause all resources in all nodes participating in the connection to be deallocated, including connection IDs, link transmit time, and internal memory buffers.

If an intermediate node cannot find the connection that is to be closed (it may have timed out at the node), the Forward_Close request shall still be forwarded to downstream nodes or the target application (via the CM_close_indication).

NOTE The Forward_Close is always forwarded to allow downstream nodes or the target application to release any resources that were allocated for the connection.

4.1.5.5.2 Format of Forward_Close request

The format of the Forward_Close request shall be of the form shown in Table 24.

Table 24 – Forward_Close request format

Parameter name	Format
connection_priority/tick time	SWORD
connection_timeout	USINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
connection_path_size	USINT
Reserved	USINT
connection_path	Padded EPATH

4.1.5.5.3 Format of Forward_Close response if success

The Forward_Close response shall be sent between Connection Managers in response to a Forward_Close request, and shall contain the status of the close request. If the status parameter of the CM_close_response is zero (no error), the format of the Forward_Close response TPDU shall be of the form shown in Table 25.

Table 25 – Forward_Close_Good response format

Parameter name	Format
connection serial number	UINT
vendor ID	UINT
originator serial number	UDINT
application_reply_size; in 16-bit words	USINT
Reserved	USINT
application_reply[]	UINT

A successful `Forward_Close` response shall be returned when the close has been acknowledged by all the nodes from this point in the path to the end of the path. The response shall indicate the `connection_serial_number`, `vendor_ID`, and `originator_serial_number` of the closed connection. Once the response indicating success has been received, the connection shall be closed from this point in the path to the end. The target application may optionally pass back application specific information in the successful close response. If no `application_reply` information is contained in the service the `application_reply_size` shall be zero.

4.1.5.5.4 Format of `Forward_Close` response if failure

If the `status` parameter of the `CM_close_response` is not zero (error), the format of the `Forward_Close` response shall be of the form shown in Table 26.

Table 26 – `Forward_Close_Bad` response format

Parameter name	Format
<code>connection_serial_number</code>	UINT
<code>vendor_ID</code>	UINT
<code>originator_serial_number</code>	UDINT
<code>remaining_path_size</code>	USINT
<code>reserved</code>	USINT

The object specific status words shall contain information about the reason for the failure. The `remaining_path_size` shall contain the path size from the point at which the close connection failed.

In the failure response, the `remaining_path_size` shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node shall return either the “pre-stripped” size or 0 for the `remaining_path_size`.

4.1.5.6 Unconnected_Send

4.1.5.6.1 Format of `Unconnected_Send` request

The `Unconnected_Send` service shall allow an application to send a message to a device without first setting up a connection. The `Unconnected_Send` service shall use the Connection Manager object in each intermediate node to forward the message and to remember the return path. The UCMM of each link shall be used to forward the request from Connection Manager to Connection Manager just as it is for the `Forward_Open` service; however, no connection shall be built. The `Unconnected_Send` service shall be sent to the local Connection Manager and shall be sent between intermediate nodes. When an intermediate node removes the last port segment, the message shall be formatted as a UCMM message and sent to the port and link address of the last segment.

NOTE The target node never sees the `Unconnected_Send` service but only a standard message arriving via the UCMM.

If the message contains an odd number of octets, a pad octet shall be appended to the end of the message allowing the remaining fields to be aligned on a 16-bit boundary. The format of the `Unconnected_Send` request shall be of the form of the form shown in Table 27.

Table 27 – Unconnected_Send request format

Parameter name	Format
priority/tick time	SWORD
conn time-out ticks	USINT
Message_size; in octets, not including pad	UINT
Message Router PDU ; if message_router_PDU is odd size, include pad octet	MR_Request
pad	USINT
path_size	USINT
Reserved	USINT
path	Padded EPATH

The message_size shall be the size of the message_router_PDU in octets without the pad. The reserved field shall be set to zero. The path_size shall be the number of 16-bit words in the path field.

4.1.5.6.2 Unconnected_Send response

The Unconnected_Send response shall be generated by the last intermediate node from the UCMM response generated by the target node or by an intermediate node as the result of a UCMM time-out, a problem with the embedded message, or a problem with the Unconnected Service Request itself. The packet shall be routed from intermediate node to intermediate node using the information stored when the Unconnected_Send request was processed. The response shall contain a header with status information about the request and a variable length response generated by the target node.

4.1.5.6.3 Format of Unconnected_Send response if success

This format shall be used when an unconnected message response is received. The application response shall be the format of the Message response from the target and shall contain error codes that resulted from the execution of the message by the application at the target node. The structure of the response is shown in Table 28.

Table 28 – Unconnected_Send_Good response format

Parameter name	Format
application_reply []	OCTET

4.1.5.6.4 Format of Unconnected_Send response if failure

This format shall be used if the originating node or any intermediate node found a problem in the message format, message parameters, or a UCMM timed-out due to no acknowledge or no response received. A UCMM time-out may be the result of

- the target device not being connected or powered up. (link specific time-out);
- an intermediate device not being connected or powered up. (link specific time-out);
- a target device being out of UCMM buffers or overloaded (Flow Control). (link specific time-out);
- an intermediate device being out of UCMM buffers or overloaded (Flow Control). (link specific time-out);
- an invalid path segment type (logical or network segment);
- an invalid port segment address (link or port address invalid).

Since the Unconnected message system does not implement any flow control, the failure of an Unconnected_Send message need not mean a device is not present in the system. The format of the Unconnected_Send response service for a failure shall be of the form shown in Table 29.

Table 29 – Unconnected_Send_Bad response format

Parameter name	Format
remaining_path_size	USINT
pad	USINT

In the failure response, the remaining Connection Path Size shall be the “pre-stripped” size. This shall be the size of the path when the node first receives the request and has not yet started processing it. A target node may return either the “pre-stripped” size or 0 for the remaining Connection Path Size.

4.1.5.7 Get_Connection_Data

4.1.5.7.1 General

This service shall return the parameters associated with a specified connection number. The connection number may be different from device to device even for the same connection. The connection number corresponds to the offset into the Connection Manager attribute that enumerates the status of the connections.

NOTE This service may be used for network diagnostics.

4.1.5.7.2 Format of Get_Connection_Data request

The format of the Get_Connection_Data request shall be of the form shown in Table 30.

Table 30 – Get_Connection_Data request format

Parameter name	Format
connection_number	UINT

4.1.5.7.3 Format of Get_Connection_Data response

The format of the Get_Connection_Data response TPDU shall be of the form shown in Table 31.

Table 31 – Get_Connection_Data response format

Parameter name	Format
connection_number	UINT
connection_state	UINT
originator_port	UINT
target_port	UINT
connection_serial_number	UINT
vendor_ID	UINT
serial_number	UDINT
originator_O2T_CID	UDINT
target_O2T_CID	UDINT
O2T_connection timeout multiplier	USINT
reserved1[3]	USINT
originator_O2T_CM_RPI	UDINT
originator_O2T_CM_API	UDINT
originator_T2O_CID	UDINT
target_T2O_CID	UDINT
T2O_connection timeout multiplier	USINT
reserved2[3]	USINT
originator_T2O_CM_RPI	UDINT
originator_T2O_CM_API	UDINT

4.1.5.8 Search_Connection_Data

4.1.5.8.1 General

This service shall return the parameters associated with a specified connection within a device identified by vendor and serial number .

NOTE This service may be used for network diagnostics.

4.1.5.8.2 Format of Search_Connection_Data request

The format of the Search_Connection_Data request shall be of the form shown in Table 32.

Table 32 – Search_Connection_Data request format

Parameter name	Format
connection_serial_number	UINT
vendor_ID	UINT
serial_number	UDINT

4.1.5.8.3 Format of Search_Connection_Data response

The format of the Search_Connection_Data response shall be the same as the response from the Get_Connection_Data service (see 4.1.5.7).

4.1.5.9 Get_Object_Owner

4.1.5.9.1 General

The Get_Object_Owner service of the Connection Manager shall return data about the connection(s) that own(s) a particular object. It shall be implemented in any device that accepts redundant connections.

4.1.5.9.2 Format of Get_Object_Owner request

The format of the Get_Object_Owner request TPDU shall be of the form shown in Table 33:

Table 33 – Get_Object_Owner request format

Parameter name	Format
reserved	USINT
path_size	USINT
path[]	UINT

The reserved field shall be set to zero.

4.1.5.9.3 Format of Get_Object_Owner response

The format of the Get_Object_Owner response TPDU shall be of the form shown in Table 34.

Table 34 – Forward_Open_Good response format

Parameter name	Format
number_of_connections	USINT
number_claiming_ownership	USINT
number_ready_for_ownership	USINT
last_action	USINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT

4.1.6 CM PDU components

4.1.6.1 Network connection parameters

4.1.6.1.1 Format

Network connection parameters shall be provided as a single 16-bit word that contains five fields, shown in Figure 4. The fields within the 16-bit word shall indicate

- the type of network connection desired;
- whether the buffer size is variable or fixed;
- the priority of the connection;
- the size of the connection buffer required.

The size of the connection buffer shall include any transport header.

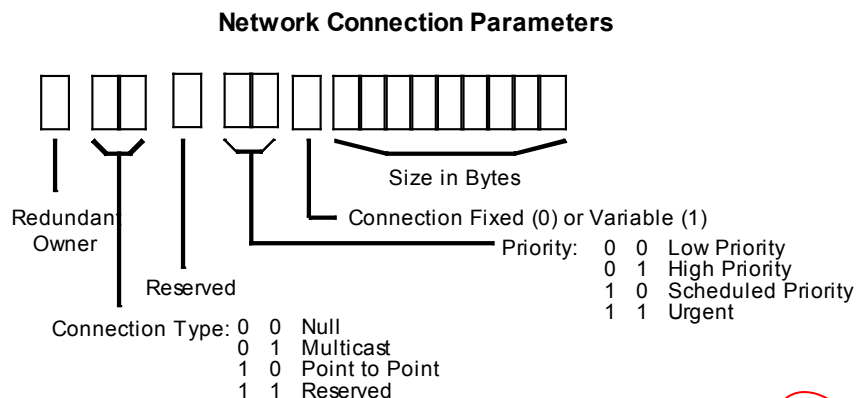


Figure 4 – Network connection parameters

4.1.6.1.2 Connection type

The connection type shall be one of NULL, MULTIPOINT, or POINT2POINT. The NULL type shall indicate that no network connection is required, while MULTIPOINT and POINT2POINT refer to the network connection types as defined in IEC 61158-5-2:2010. The NULL connection type may be used to reconfigure a connection.

4.1.6.1.3 Priority

Priority shall be one of LOW, HIGH, and SCHEDULED as specified in IEC 61158-4-2:2010.

NOTE SCHEDULED data is prioritized on a link wide basis; however, the two unscheduled priorities (LOW and HIGH) are arbitrated independently at each node. One node may transmit a LOW priority packet while another node has HIGH priority data to send.

4.1.6.1.4 Connection fixed/variable

The connection can be set up to use fixed or variable sized connections. With a fixed size connection, each transmission on the connection shall use the same size buffer. Otherwise, either a buffer overrun or underrun condition shall occur. With a variable sized connection, each transmission on the connection may send a variable amount of data up to the maximum size, which shall be specified when the connection is opened. Buffer underruns shall not be reported, but a buffer overrun can still occur if too much data is sent.

4.1.6.1.5 Connection size

The network connection size shall be the size of the buffer required for the connection, which includes the data and any transport header. For a variable sized connection, the size shall be the maximum size of the buffer for any transfer. The actual size of the transfer for a variable connection shall be equal to or less than the size specified for the network connection. The maximum buffer size shall be dependent on the links that the connection traverses.

4.1.6.1.6 Redundant owner

The redundant owner bit in the O⇒T direction shall be set (= 1) to indicate that more than one owner may be permitted to make a connection simultaneously. The bit shall be clear (= 0) to indicate an exclusive-owner, input only or listen-only connection.

4.1.6.1.7 Reserved parameters

Reserved bits shall be clear (= 0).

4.1.6.2 Packet interval

4.1.6.2.1 Requested packet interval (CM_RPI)

The requested packet interval shall be the requested time between packets in microseconds. The format of the CM_RPI shall be a 32-bit integer in microseconds.

4.1.6.2.2 Actual packet interval (CM_API)

The actual packet interval shall be the actual time between packets in microseconds. The format of the CM_API shall be a 32-bit integer in microseconds.

4.1.6.2.3 Usage

The requested packet interval shall be the time between packets requested by the receiving device. The value shall be used to allocate link transmit time at each of the producing nodes. The allocation of link transmit time may have to be adjusted when the actual packet rate or actual packet interval is returned, since it is possible for the two values to differ. The path time-out value at each of the intermediate and target nodes shall also be set to the connection time-out multiplier times the CM_API. The CM_RPI is therefore required for all connections.

4.1.6.2.4 Function of priorities

Scheduled Priority

For scheduled priority, the CM_RPI shall be the packet rate of the repetitive data. On links that support allocation of link transmit time, link transmit time shall be reserved for this packet. For scheduled priority, the data shall also be restricted to the specified packet rate, which means that if data arrives at an intermediate node faster than the specified packet rate, the node shall filter the packets to the specified rate. Since each node's scheduled priority update rate is in discrete quanta, the Actual Packet Interval (CM_API) may be smaller (more rapid) than the CM_RPI. The path time-out value shall be set to the Connection Time-out multiplier times the CM_API.

High Priority

For high priority, the CM_RPI shall be used to set the path time-out in the intermediate and target nodes. The CM_RPI shall therefore be set to the slowest packet rate expected, which shall preclude having the connection close due to a path time-out. The longer the path time-out, the longer the time required to reclaim resources in the intermediate nodes as a result of faults in the network. Since the high priority is not quantized at any of the nodes, the CM_API shall equal the CM_RPI. To maintain consistency, however, the time-out value shall again be set to the Connection Time-out multiplier times the CM_API.

Low Priority

For low priority, the CM_RPI shall be used to set the path time-out in the intermediate and target nodes. The CM_RPI shall therefore be set to the slowest packet rate expected, which shall preclude having the connection close due to a path time-out. The longer the path time-out, the longer the time required to reclaim resources in the intermediate nodes as a result of faults in the network. Since the low priority is not quantized at any of the nodes, the CM_API shall equal the CM_RPI. To maintain consistency, however, the time-out value shall again be set to the Connection Time-out multiplier times the CM_API.

4.1.6.3 Connection time-out multiplier

The Connection Time-out Multiplier specifies the multiplier applied to the CM_RPI to obtain the connection time-out value. Device shall stop transmitting on a connection whenever the connection times out even if the pending close has been sent. The multiplier shall be as represented by Table 35.

Table 35 – Time-out multiplier

Value	Multiplier	Notes
0	X 4	Default
1	X 8	
2	X 16	
3	X 32	
4	X 64	
5	X 128	
6	X 256	
7	X 512	
8 – 255	reserved	

4.1.6.4 Connection request priority and tick time

Two values are packed into this field. The connection request priority determines the priority of the UCMM messages used to establish the connection and has no correlation with the priority of the connection being made. Although Low priority is used for most connection establishments, High priority may be used for those special circumstances which require a connection be made quickly. The use of low priority was chosen to avoid congestion with time-critical messaging. UCMM messages can not use scheduled priority.

The tick time shall be used in conjunction with the connection request time-out ticks value. This value determines the time between "ticks" in the latter field. Thus, if the tick time is 1ms., then a connection request time-out value of 5 would translate into 5 ms. If the tick time was 2 ms, then a connection request time-out value of 5 would translate into 10 ms.

The format of the Connection Request Priority/Tick Time shall be as shown in Figure 5.

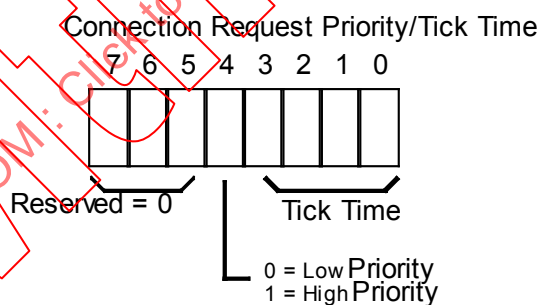
**Figure 5 – Time tick**

Table 36 shall determine the time between ticks.

Table 36 – Time tick units

Tick time		
Tick time (binary)	Time per tick	Max. time
0000	1 ms	255 ms
0001	2 ms	510 ms
0010	4 ms	1 020 ms
0011	8 ms	2 040 ms
0100	16 ms	4 080 ms

Tick time		
Tick time (binary)	Time per tick	Max. time
0101	32 ms	8 160 ms
0110	64 ms	16 320 ms
0111	128 ms	32 640 ms
1000	256 ms	65 280 ms
1001	512 ms	130 560 ms
1010	1 024 ms	261 120 ms
1011	2 048 ms	522 240 ms
1100	4 096 ms	1 044 480 ms
1101	8 192 ms	2 088 960 ms
1110	16 384 ms	4 177 920 ms
1111	32 768 ms	8 355 840 ms

4.1.6.5 Connection request time-out ticks

The connection request time-out ticks shall be used to specify the amount of time the originating application shall wait for the connection to be established. Each hop of the connection has a link specific time-out value implemented by the UCMM, and after the automatic retries have been exhausted without an acknowledge the UCMM shall generate a time-out error. Therefore if a device in the path shall be either not present or too busy to acknowledge a UCMM request, a UCMM time-out shall occur as fast as would be possible. The only time the connection request time-out period would be encountered shall be if the target node acknowledges the connection request but fails to respond or if the connection request shall be delivered but a failure in the network prevents the response from being returned. The connection request time-out shall be not the path time-out value derived from the CM_RPI and used to close a connection after it shall be established. The connection request time-out shall be specified by a combination of the connection request time-out ticks multiplied by the time per tick which shall be determined by the tick time value.

Since a particular connection request time-out can be expressed in several different formats, the rule shall be to **always pack it such that the tick time value shall be minimized**. This allows the best granularity for decreasing the time-out as the message is passed through hops. This also means that the tick time value may change as the message is passed through multiple hops. As an example, 300 ms may be encoded as:

1. 0001 10010110 -> Time per Tick = 2 ms, Ticks = 150 (right)
2. 0010 01001011 -> Time per Tick = 4 ms, Ticks = 75 (wrong)

The connection request time-out value shall be dependent on the number of port segments in the connection path, the processing time at the target node, and the congestion of the intermediate networks. The connection request time-out can be a relatively large value since the full connection request time-out shall only occur in a few rare cases. The only time the full connection request time-out would occur would be if:

- the connection request was successfully delivered to the target node and the node failed to generate a response. This would usually be caused by a firmware error in the device.
- the connection request was successfully delivered to the target node and the response was discarded due to congestion at an intermediate node.
- the connection request was successfully delivered to the target node and the target node or one of the intermediate nodes was removed or powered down before the response was returned.

The UCMM shall be used to forward the connection request to the next node in the path. The UCMM sends the request and then waits a link specific time for an acknowledgement or response, if no acknowledgement or response shall be received before the link specific time-out, the request shall be retried. After a link specific number of retries (typically three), the UCMM shall signal a time-out on the connection request. The Connection Manager which generated the request shall return an error response to the connection request. Therefore the connection request shall time-out as soon as possible for the most common system faults

- missing or powered down devices in the path;
- congestion at a device in the path.

To aid in debugging systems, each hop in the path shall decrement the time-out by about 80 ms before sending it out in the Forward_Open to the next device in the path. Each intermediate device (router) shall round the connection time-out value that shall be decremented such that a minimum of 1 tick shall be subtracted. This shall enable the last device before the failure to time-out first and return information about how far the connection was made before the failure. This information can be used to help in debugging problems in a system. A typical system function is shown in Figure 6.

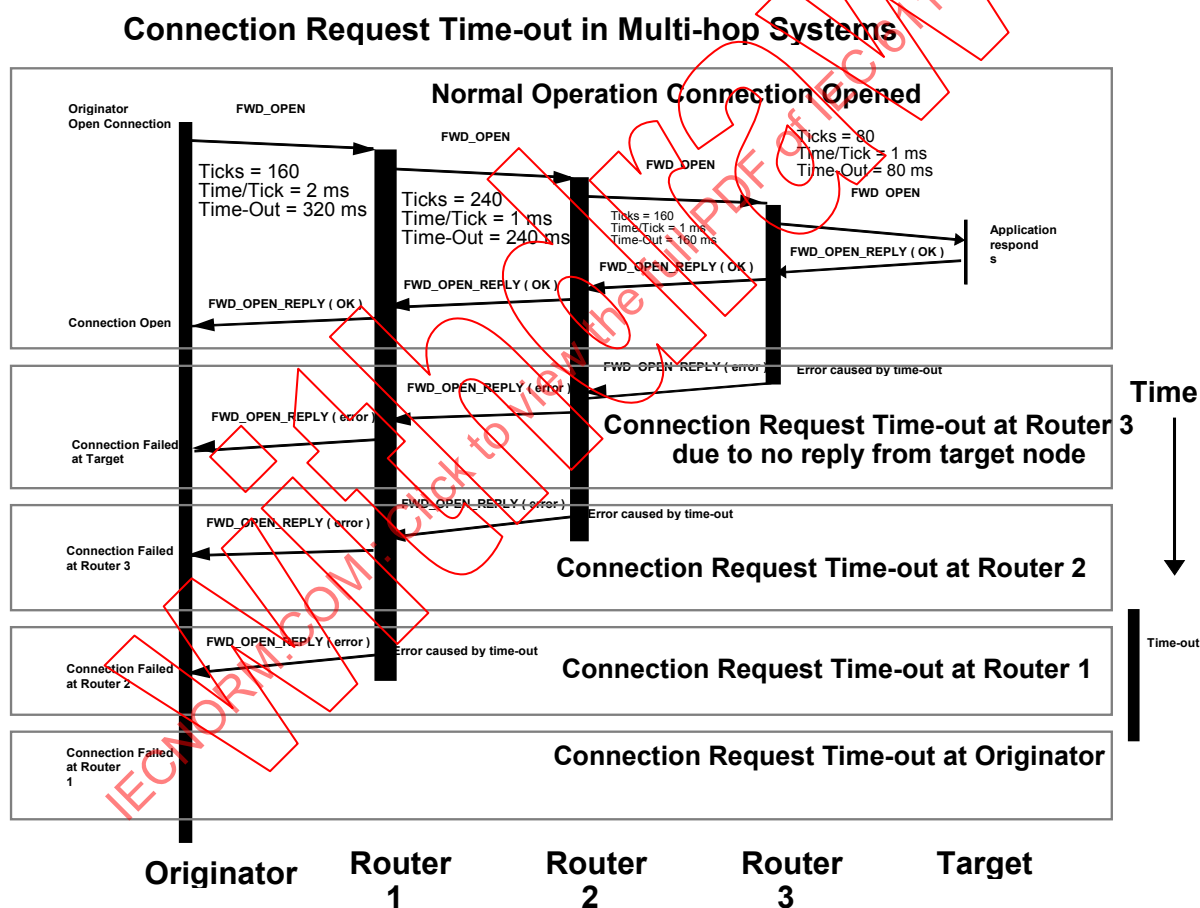


Figure 6 – Connection establishment time-out

In Figure 6 a connection is requested over a path containing 4 hops. The Connection request Time-out is specified as 320 ms by the originator of the connection. In the first case the connection is completed and the calculated time-out values are shown at each hop in the path. The second case is where the last device receives an acknowledgement but does not receive a response and times out. It returns a time-out error to the originator. Since the time-out value decreases for each hop, the time-out error shall be returned to the originator before the originator times out.

The other cases show time-outs at the other devices, and in each case the time-out error shall be returned to the originator before the originator itself times out. The last case is where no response is received by the originator and the connection open process is timed out. The reason to decrease the time-out at each node is so additional error information indicating where in the path the time-out occurred can be sent. Remember that if any of the devices do not receive an acknowledgement, the connection request shall immediately be timed out and an error response shall be generated.

4.1.6.6 Connection serial number

The connection serial number shall be a unique 16-bit value selected by the Connection Manager at the originator of the connection. The originator shall make sure that the 16-bit value is unique for the device. There shall be no other significance placed on the number by any other nodes in the connection path. The connection serial numbers shall be unique but do not have to be sequential. For example, an operator interface may have a large number of connections open at the same time, each with a unique number. The same values could be repeated at other operator interface stations. A possible implementation would be to have a connection list which points to the descriptor for each connection, and the connection serial number could be the index into the table.

4.1.6.7 Vendor ID

The vendor number shall be a unique number assigned to the various vendors of products. Each vendor has a unique number assigned.

NOTE Vendor IDs are assigned by ODVA, Inc.

4.1.6.8 Originator serial number

The originator serial number shall be a unique 32-bit value that is assigned to a device at the time of manufacture. This value shall be guaranteed to be unique for all devices manufactured by the same vendor. No significance shall be attached to the number. The combination of Vendor ID and Originator Serial Number shall be unique throughout the entire system.

4.1.6.9 Connection number

The connection number shall be a 16-bit value that is assigned by the Connection Manager when a connection is opened. This value allows other nodes to obtain connection data from the Connection Manager. This number shall not be confused with the Connection Serial Number.

4.1.6.10 Connection path size

The connection path size shall be the length of the connection path in 16-bit words. The length of the connection path varies during the connection process, since each node in the connection path removes the current port segment and forwards only the remaining path segments to the next node.

4.1.6.11 Connection path

Refer to Path description in 4.1.8.9.

4.1.6.12 Connection remaining path

Refer to Path description in 4.1.8.9.

4.1.6.13 Network connection ID

The Network Connection ID shall be a 32-bit value built as follows :

- Bits 31-24 shall be reserved and shall be set to zero;
- Bits 23-16 shall contain a node MAC ID depending on selected connection type

- Bits 15-0 shall contain the 16-bit connection number.

Table 37 shows options for selection of Connection ID.

Table 37 – Selection of connection ID

Connection type	MAC ID	Connection number
Multipoint	MAC of Producer	Unique by Device
Point to Point	MAC of Consumer	Unique by Device

The Network Connection ID shall be link specific and shall not be related to the connection serial number, which is connection specific and the same over all the links. The fields of the Network Connection ID shall be used to set the screening mechanism for the specified link.

The generic tag for the Lpackets exchanged over a connection and described in IEC 61158-4-2:2010, shall be built as follows, based on the contents of the CID fields:

- the first transmitted octet of the generic tag shall contain the MAC ID;
- the second transmitted octet of the generic tag shall contain the least significant octet of the connection number;
- the third transmitted octet of the generic tag shall contain the most significant octet of the connection number.

To guarantee that connection IDs are unique on a link the following rules shall be used when choosing CIDs:

- the producer shall choose the CID for a multipoint transport;
- the consumer shall choose the CID for a point-to-point transport;
- on a CP 2/1 link, the node responsible for the choice of the CID (either producer or consumer) shall insert its MAC ID into bits 23-16 of the CID in the Forward_Open;
- a multipoint CID shall not be reused until all connections associated with the CID have been closed or timed out;
- receiving an I'm alive fixed tag packet from a node shall immediately close any connections with that device.

4.1.6.14 Transport class and trigger

The transport class and trigger specify the type of transport required for the connection. This information shall not be used by the Connection Manager but passed on to the application. The application shall determine if the transport type is supported and if an instance of the required transport is available. The octet that encode the transport class and trigger shall be of the following form shown in Table 38.

Table 38 – Transport class, trigger and Is_Server format

TransportClass (4 bits)	TriggerMode (3 bits)	Is_Server (1 bit)
NULL = 0 DUPLICATE_DETECT = 1 ACKNOWLEDGED = 2 VERIFIED = 3 NONBLOCKING = 4 FRAGMENTING = 5 MULTIPOINT_FRAG = 6	CYCLIC = 0 CHANGE_OF_STATE = 1 APPLICATION = 2	IS_NOT_SERVER = 0 IS_SERVER = 1

The least significant 4 bits, *class*, shall determine which transport type is specified and shall be in the range 0 to 6. The *class* values 7 through 15 shall be reserved. Bits 4-6, called *trigger*, shall determine what event causes the production of data on the connection and

shall be in the range 0 to 2. The `trigger` values 3 through 7 shall be reserved. For transports that need to specify the target as a server, the `is_server` field shall be 1; otherwise, it shall be 0. For transport classes 0 and 1, the `is_server` field shall be ignored. Bit 7 shall be the `is_server` field;

See IEC 61158-5-2:2010 for details on the behaviour for the different combinations of `is_server`, `class`, and `trigger`, `CM_RPI`, and `CM_API`.

4.1.7 MR headers

All request packets delivered to the Message Router, either from a connection or from the UCMM, shall have a header of the form shown in Table 39.

Table 39 – MR_Request_Header format

Parameter	Format
Service	USINT
path_size	USINT
path[]	UINT

The first octet, `service`, shall be directly delivered to the application object and shall be in the range 1 through 127. The `path_size` shall determine the number of 16-bit words in the `path` array. The `path` shall contain addressing information for the packet and shall be padded.

Response packets from the Message Router shall have a header of the form shown in Table 40.

Table 40 – MR_Response_Header format

Parameter	Format
Service	USINT
Reserved	USINT
general_status	USINT
Extended_status_size	USINT
Extended_status[]	WORD

The `service` field shall be the value of the `MR_Request_Header.service` field with its most significant bit set. If `MR_Request_Header.service` equals 0x05, the `MR_Response_Header.service` shall equal 0x85. The `general_status` field shall be filled using the `Status Code` parameter of the service response. Likewise the `extended_status`, `extended_status_size`, and `response` shall be filled using the `Extended Status` and other response parameters from the service response, respectively. The `response_size` shall not be explicitly included in the `MR_Response_Header`; it shall be implied by number of octets in the `MR_Response_Header`.

4.1.8 OM_Service_PDU

4.1.8.1 General syntax

4.1.8.1.1 Get_Attribute_All

The `Get_Attribute_All_RequestPDU` body is empty.

The `Get_Attribute_All_ResponsePDU` body shall be as specified in Table 41.

Table 41 – Structure of Get_Attribute_All_ResponsePDU body

Name	Data type
Attribute Data	Object/class attribute-specific

4.1.8.1.2 Set_Attribute_All

The Set_Attribute_All_RequestPDU body shall be as specified in Table 42.

Table 42 – Structure of Set_Attribute_All_RequestPDU body

Name	Data type
Attribute Data	Object/class attribute-specific

The Get_Attribute_All_ResponsePDU body is empty.

4.1.8.1.3 Get_Attribute_List

The Get_Attribute_List_RequestPDU body shall be as specified in Table 43.

Table 43 – Structure of Get_Attribute_List_RequestPDU body

Name	Data type
Attribute Count	UINT
Attribute List	Array of UINT's

The Get_Attribute_List_ResponsePDU body shall be as specified in Table 44.

Table 44 – Structure of Get_Attribute_List_ResponsePDU body

Name	Data type	Semantics of values
Attribute Count	UINT	
Attribute Data	Array of STRUCT	
Attribute Identifier	UINT	
Attribute Status	UINT	Values 0-255 are reserved to mirror common service status codes. Values 256-65 535 are available for object/class attribute-specific errors.
Attribute Value	Object/class attribute- specific	

4.1.8.1.4 Set_Attribute_List

The Set_Attribute_List_RequestPDU body shall be as specified in Table 45.

Table 45 – Structure of Set_Attribute_List_RequestPDU body

Name	Data type
Attribute Count	UINT
Attribute Data	Array of STRUCT
Attribute Identifier	UINT
Attribute Value	Object/class attribute- specific

The Set_Attribute_List_ResponsePDU body shall be as specified in Table 46.

Table 46 – Structure of Set_Attribute_List_ResponsePDU body

Name	Data type	Semantics of values
Attribute Count	UINT	
Attribute Status List	Array of STRUCT	
Attribute Identifier	UINT	
Attribute Status	UINT	Values 0-255 are reserved to mirror common service status codes. Values 256-65 535 are available for object/class attribute-specific errors

4.1.8.1.5 Reset

The Reset_RequestPDU body shall be as specified in Table 47, if the **optional** “Object Specific Data” service parameter of the Reset request is specified. Else it shall be empty.

Table 47 – Structure of Reset_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Reset_ResponsePDU body shall be as specified in Table 48, if the **optional** “Object Specific Data” service parameter of the Reset response is specified. Else it shall be empty.

Table 48 – Structure of Reset_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.6 Start

The Start_RequestPDU body shall be as specified in Table 49, if the **optional** “Object Specific Data” service parameter of the Start request is specified. Else it shall be empty.

Table 49 – Structure of Start_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Start_ResponsePDU body shall be as specified in Table 50, if the **optional** “Object Specific Data” service parameter of the Start response is specified. Else it shall be empty.

Table 50 – Structure of Start_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.7 Stop

The Stop_RequestPDU body shall be as specified in Table 51, if the **optional** “Object Specific Data” service parameter of the Stop request is specified. Else it shall be empty.

Table 51 – Structure of Stop_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Stop_ResponsePDU body shall be as specified in Table 52, if the **optional** “Object Specific Data” service parameter of the Stop response is specified. Else it shall be empty.

Table 52 – Structure of Stop_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.8 Create

The Create_RequestPDU body shall be as specified in Table 53, if the **optional** “Object Specific Data” service parameter of the Create request is specified. Else it shall be empty.

Table 53 – Structure of Create_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Create_ResponsePDU body shall be as specified in Table 54, if the **optional** “Object Specific Data” service parameter of the Create response is specified. Else it shall be empty.

Table 54 – Structure of Create_ResponsePDU body

Name	Data type
Instance ID	UINT
Object Specific Data	Object/class service- specific

4.1.8.1.9 Delete

The Delete_RequestPDU body shall be as specified in Table 55, if the **optional** “Object Specific Data” service parameter of the Delete request is specified. Else it shall be empty.

Table 55 – Structure of Delete_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Delete_ResponsePDU body shall be as specified in Table 56, if the **optional** “Object Specific Data” service parameter of the Delete response is specified. Else it shall be empty.

Table 56 – Structure of Delete_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.10 Get_Attribute_Single

The Get_Attribute_Single_RequestPDU body is empty.

The Get_Attribute_Single_ResponsePDU body shall be as specified in Table 57.

Table 57 – Structure of Get_Attribute_Single_ResponsePDU body

Name	Data type
Attribute Data	Object/class attribute-specific

4.1.8.1.11 Set_Attribute single

The Set_Attribute_Single_RequestPDU body shall be as specified in Table 58.

Table 58 – Structure of Set_Attribute_Single_RequestPDU body

Name	Data type
Attribute Data	Object/class attribute-specific

The Set_Attribute_Single_ResponsePDU body shall be as specified in Table 59, if the **optional** “Object Specific Data” service parameter of the Set_Attribute_Single response is specified. Else it shall be empty.

Table 59 – Structure of Set_Attribute_Single_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service-specific

4.1.8.1.12 Find_Next_Object_Instance

The Find_Next_Object_Instance_RequestPDU body shall be as specified in Table 60.

Table 60 – Structure of Find_Next_Object_Instance_RequestPDU body

Name	Data type
Maximum Returned Values	USINT

The Find_Next_Object_Instance_ResponsePDU body shall be as specified in Table 61.

Table 61 – Structure of Find_Next_Object_Instance_ResponsePDU body

Name	Data type
Number Of List Members	USINT
Instance ID List	Array of UINT

4.1.8.1.13 NOP

The NOP_RequestPDU body is empty.

The NOP_ResponsePDU body is empty.

4.1.8.1.14 Apply_Attributes

The Apply_Attributes_RequestPDU body shall be as specified in Table 62, if the **optional** “Object Specific Data” service parameter of the Apply_Attributes request is specified. Else it shall be empty.

Table 62 – Structure of Apply_Attributes_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Apply_Attributes_ResponsePDU body shall be as specified in Table 63, if the **optional** “Object Specific Data” service parameter of the Apply_Attributes response is specified. Else it shall be empty.

Table 63 – Structure of Apply_Attributes_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.15 Save

The Save_RequestPDU body shall be as specified in Table 64, if the **optional** “Object Specific Data” service parameter of the Save request is specified. Else it shall be empty.

Table 64 – Structure of Save_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Save_ResponsePDU body shall be as specified in Table 65, if the **optional** “Object Specific Data” service parameter of the Save response is specified. Else it shall be empty.

Table 65 – Structure of Save_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.16 Restore

The Restore_RequestPDU body shall be as specified in Table 66, if the **optional** “Object Specific Data” service parameter of the Restore request is specified. Else it shall be empty.

Table 66 – Structure of Restore_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Restore_ResponsePDU body shall be as specified in Table 67, if the **optional** “Object Specific Data” service parameter of the Restore response is specified. Else it shall be empty.

Table 67 – Structure of Restore_ResponsePDU body

Name	Data type
Object Specific Data	Object/class service- specific

4.1.8.1.17 Group_Sync

The Group_Sync_RequestPDU body shall be as specified in Table 68.

Table 68 – Structure of Group_Sync_RequestPDU body

Name	Data type
Object Specific Data	Object/class service- specific

The Group_Sync_ResponsePDU body shall be as specified in Table 69.

Table 69 – Structure of Group_Sync_ResponsePDU body

Name	Data type	Semantics of values
IsSynchronized	BOOL	Indicates if object is synchronized to the PTP Time Master 0 = Not synchronized 1 = Is synchronized
Object Specific Data	Object/class service- specific	Object/class service- specific

4.1.8.2 Identity object specific syntax elements

4.1.8.2.1 Attributes

4.1.8.2.1.1 Class attributes

The format of the Identity object class attributes shall be as specified in Table 70.

Table 70 – Identity object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
2	Max Instance	UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.2.1.2 Instance attributes

The format of the Identity object instance attributes is as specified in Table 71.

Table 71 – Identity object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Vendor ID	UINT	The value zero shall not be used.
2	Device Type	UINT	A listing of the Device Type ranges can be found in 4.1.10.2.5.
3	Product Code	UINT	The value zero is not valid.
4	Revision	STRUCT of	The value zero shall not be valid for either the Major and Minor Revision fields of a compliant product. A damaged product may return zero to indicate unknown revision if its storage of revision become corrupt.
	Major Revision	USINT	The Major Revision attribute shall be limited to 7 bits. The eighth bit is reserved and shall be zero.
	Minor Revision	USINT	
5	Status	WORD	Bit definitions are described in Table 72.

Attribute ID	Name	Data type	Semantics of values
6	Serial Number	UDINT	
7	Product Name	SHORT_STRING	The maximum number of 8-bit characters in this string shall be 32. Each of the characters shall be in the range 0x20 to 0x7E.
8	State	USINT	Present state of the device as represented by the state transition diagram: 0 = Nonexistent 1 = Device Self Testing 2 = Standby 3 = Operational 4 = Major Recoverable Fault 5 = Major Unrecoverable Fault 6 – 254 = Reserved 255 = Default for Get_Attributes_All service
9	Configuration Consistency Value	UINT	Contents identify configuration of device
10	Heartbeat Interval	USINT	The nominal interval between heartbeat messages in seconds.
11	Active Language	STRUCT of	Currently active language for the device.
	Language1	USINT	The language1 field from the STRING1 data type.
	Language2	USINT	The language2 field from the STRING1 data type.
	Language3	USINT	The language3 field from the STRING1 data type.
12	Supported Language	ARRAY of STRUCT of	List of languages supported by character strings of data type STRING1 within the device.
	Language1	USINT	The language1 field from the STRING1 data type.
	Language2	USINT	The language2 field from the STRING1 data type.
	Language3	USINT	The language3 field from the STRING1 data type.
13	International Product Name	STRING1	Names of the product in various languages
14	Semaphore	STRUCT of	Access Synchronization Semaphore
	Vendor Number	UINT	Default: All zero values
	Client Serial Number	UDINT	
	Semaphore Timer	ITIME	Timer valid range 100 thru 32 767
15	Assigned_Name	STRING1	Use assigned name
16	Assigned_Description	STRING1	User assigned description
17	Geographic_Location	STRING1	User assigned location

Table 72 – Identity object bit definitions for status instance attribute

Bit(s)	Called	Definition
0	Owned	1 for TRUE, 0 for FALSE
1		Reserved, set to 0
2	Configured	1 for TRUE, 0 for FALSE
3		Reserved, set to 0
4,5,6,7		See Table 73
8	Minor Recoverable Fault	1 for TRUE, 0 for FALSE
9	Minor Unrecoverable Fault	1 for TRUE, 0 for FALSE
10	Major Recoverable Fault	1 for TRUE, 0 for FALSE
11	Major Unrecoverable Fault	1 for TRUE, 0 for FALSE

Bit(s)	Called	Definition
12, 13		Reserved, shall be set to 0
14, 15		Reserved, shall be set to 0

The values of the status bits 4 to 7 shall be as shown in Table 73.

Table 73 – Bits 4 – 7 of status instance attribute

Bits 4, 5, 6, 7:	Meaning
0 0 0 0	Self-Testing (power-up) or State Unknown
0 0 0 1	Firmware Update in progress
0 0 1 0	Communication Fault (lost connection)
0 0 1 1	Unkeyed, Awaiting Connection
0 1 0 0	Non-Volatile Configuration Bad
0 1 0 1	Major Fault (see bits 10 and 11 for fault classification and LED states) Minor fault is not a state change
0 1 1 0	Connected, Active
0 1 1 1	Idle (program mode type)
1 0 0 0 1 0 0 1	reserved, shall not be used
1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 1 1 1 1 0 1 1 1 1	reserved for product specific states

4.1.8.2.2 Common services

4.1.8.2.2.1 Get_Attribute_All Response

At the **class level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 74.

Table 74 – Class level object/service specific response data of Get_Attribute_All

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Revision (low octet) Default = 1							
	1	Revision (high octet) Default = 0							
2	2	Max Instance (low octet) Default = 1							
	3	Max Instance (high octet) Default = 0							
6	4	Max ID Number of Class Attributes (low octet) Default = 0							
	5	Max ID Number of Class Attributes (high octet) Default = 0							
7	6	Max ID Number of Instance Attributes (low octet) Default = 0							
	7	Max ID Number of Instance Attributes (high octet) Default = 0							

At the **instance level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as defined in Table 75.

Table 75 – Instance level object/service specific response data of Get_Attribute_All

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
1	0	Vendor (low octet)							
	1	Vendor (high octet)							
2	2	Device Type (low octet)							
	3	Device Type (high octet)							
3	4	Product Code (low octet)							
	5	Product Code (high octet)							
4	6	Major Revision							
	7	Minor Revision							
5	8	Status (low octet)							
	9	Status (high octet)							
6	10	Serial Number (low octet)							
	11	Serial Number							
	12	Serial Number							
	13	Serial Number (high octet)							
7	14	Product Name length							
	15	Product Name (1 st character)							
	16	Product Name (2 nd character)							
	n	Product Name (last character)							

Because the length of the name is not known before issuing the Get_Attribute_All service request, implementers shall allow enough memory space to store a response up to 32 characters in length.

To maintain consistency, if any instance attributes are added at some later date, the instance Get_Attribute_All response shall be modified to include the attributes as defined in Table 76.

Table 76 – Modified instance level object/service specific response data of Get_Attribute_All

Attribute ID	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8	n+1	State Default = 255							
9	n+2	Space reserved for attribute 9 low octet, default = 0							
	n+3	Space reserved for attribute 9 high octet, default = 0							
10	n+4	Heartbeat Interval, Default = 0							

4.1.8.2.2.2 Reset service

The Reset common service shall have the object-specific parameter specified in Table 77.

Table 77 – Object-specific parameter for Reset

Name	Type	Semantics of values
Type	USINT	Value 0 shall emulate as closely as possible cycling power on the item the Identity object represents. This value shall be the default if this parameter is omitted. Value 1 shall return as closely as possible to the out-of-box configuration, then shall emulate cycling power as closely as possible. Value 2 shall return as closely as possible to the out-of-box configuration with the exception of communication link parameters, then emulate cycling power as closely as possible. The communication link parameters that are to be preserved are defined by each network type. Values 3-255 are reserved.

4.1.8.3 Message Router object specific syntax elements

4.1.8.3.1 Attributes

4.1.8.3.1.1 Class attributes

The Message Router object class attributes shall be as specified in Table 78.

Table 78 – Message Router object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.3.1.2 Instance attributes

The format of the Message Router object instance attributes is as specified in Table 79.

Table 79 – Message Router object instance attributes

Attribute ID	Name	Data type
1	Object_List	STRUCT of
	Number	UINT
	Classes	ARRAY of UINT
2	Number available	UINT

4.1.8.3.2 Common services

Get_Attribute_All Response

At the **class level**, the order of the attributes returned in the “Attribute Data” parameter of the Get_Attribute_All response shall be as specified in Table 80. Default values for all unsupported class attributes shall be as shown in that table.

Table 80 – Class level object/service specific response data of Get_Attribute_All

Class attribute ID	Attribute name, description and default value
1	Revision default = 1
4	Optional attribute list, number of attributes default = 0
5	Optional service list, number of services default = 0
6	Max ID number of class attributes default = 0
7	Max ID number of instance attributes default = 0

Default values for all unsupported instance attributes shall be as shown in Table 81.

Table 81 – Instance level object/service specific response data of Get_Attribute_All

Instance attribute ID	Attribute name, description and default value
1	Object list number, number of supported classes default = 0
2	Number available, maximum number of connections default = 0
3	Always = 0x0000

4.1.8.4 Assembly object specific syntax elements

4.1.8.4.1 Attributes

4.1.8.4.1.1 Class attributes

The format of the Assembly object class attributes shall be as specified in Table 82.

Table 82 – Assembly object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 2
2	Max Instance	UINT	

4.1.8.4.1.2 Instance attributes

The format of the Assembly object instance attributes is as specified in Table 83.

Table 83 – Assembly object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Number of Members in List	UINT	
2	Member List	ARRAY of STRUCT	This attribute has a complex data type
	Member Data Size	UINT	Size in bits
	Member Path Size	UINT	Size in octets (0 = Empty Path)
	Member Path	Packed EPATH	Packed path to the member data. See 4.1.8.9 for definition of Path

Attribute ID	Name	Data type	Semantics of values
3	Data	ARRAY of SWORD	Contain all of the member data packed into one array. This data may contain many different data types. For efficiency it is best to keep this data word aligned by packing it on word boundaries and adding padding as needed. This can be accomplished by using "empty paths" (Member Path Size = 0)
4	Size	UINT	

4.1.8.5 Acknowledge Handler object specific syntax elements

4.1.8.5.1 Attributes

4.1.8.5.1.1 Class attributes

The format of the Acknowledge Handler object class attributes shall be as specified in Table 84.

Table 84 – Acknowledge Handler object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.5.1.2 Instance attributes

The format of the Acknowledge Handler object instance attributes is as specified in Table 85.

Table 85 – Acknowledge Handler object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Acknowledge Timer	UINT	Range 1 – 65 535 ms (0 invalid) Default = 16
2	Retry Limit	USINT	Range 0 – 255 Default = 1
3	COS Producing Connection Instance	UINT	Connection Instance ID
4	Ack List Size	SWORD	0 = Dynamic > 0 = Max number of members
5	Ack List	STRUCT of	
		SWORD	Number of members in the array
		ARRAY of UINT	List of Connection Instance IDs
6	Data with Ack Path List Size	SWORD	0 = Dynamic > 0 = Max number of members
7	Data with Ack Path List	STRUCT of	
		SWORD	Number of members in the array
		ARRAY of	
		UINT	Connection Instance ID
		USINT	Path length
		Padded EPATH	Path

4.1.8.5.2 Object specific services

4.1.8.5.2.1 Add_AckData_Path

The Add_AckData_Path_RequestPDU body shall be as specified in Table 86.

Table 86 – Structure of Add_AckData_Path_RequestPDU body

Name	Data type
Connection Instance ID	UINT
Consumer Path Size	USINT
Consumer Path	Padded EPATH

The Add_AckData_Path_ResponsePDU body is empty.

4.1.8.5.2.2 Remove_AckData_Path

The Add_AckData_Path_RequestPDU body shall be as specified in Table 87.

Table 87 – Structure of Remove_AckData_Path_RequestPDU body

Name	Data type
Connection Instance ID	UINT

The Remove_AckData_Path_ResponsePDU body is empty.

4.1.8.6 Time Sync object specific syntax elements

4.1.8.6.1 Attributes

Instance attributes

The format of the Time Sync object instance attributes shall be as specified in Table 88.

Table 88 – Time Sync object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	PTPEnable	BOOL	PTP enabled for this device: 0 = PTP not enabled 1 = PTP enabled
2	IsSynchronized	BOOL	Local clock synchronized with reference clock: 0 = local clock not synchronized 1 = local clock synchronized
3	SystemTimeMicroseconds	ULINT	Current value of local time in microseconds
4	SystemTimeNanoseconds	ULINT	Current value of local time in nanoseconds
5	OffsetFromMaster	LINT	Offset between local clock and master clock in nanoseconds
6	MaxOffsetFromMaster	ULINT	Maximum offset between local clock and master clock in nanoseconds
7	MeanPathDelayToMaster	LINT	Path delay to master
8	GrandMasterClockInfo	STRUCT of	Grandmaster Clock information
	ClockIdentity	USINT[8]	Unique identifier of the clock. See Table 89 for ClockIdentity encoding
	ClockClass	UINT	Class of the clock quality. Table 90 shows the values most likely to be used by the Time sync object. See IEC 61588:2009 for a complete list of values
	TimeAccuracy	UINT	Expected absolute accuracy of the clock relative to the PTP epoch. See Table 91 for TimeAccuracy values
	OffsetScaledLogVariance	UINT	Measure of the inherent stability properties of the clock
	CurrentUtcOffset	UINT	Current UTC offset in seconds from International Atomic Time (TAI) of the clock. As of 0 hours 1 January 2006 UTC, the offset was 33 seconds
	TimePropertyFlags	WORD	For the value of the TimePropertyFlags see Table 92
	TimeSource	UINT	For the possible TimeSource values see Table 93
	Priority1	UINT	Relative priority of the grandmaster clock to other clocks in the system. See Priority1 attribute 16
	Priority2	UINT	Relative priority of the grandmaster clock to other clocks in the system. See Priority2 attribute 17
9	ParentClockInfo	STRUCT of	Parent Clock information
	ClockIdentity	USINT[8]	Unique identifier of the clock. See Table 89 for ClockIdentity encoding
	ClockClass	UINT	Class of the clock quality. Table 90 shows the values most likely to be used by the Time sync object. See IEC 61588:2009 for a complete list of values
	Time Accuracy	UINT	Expected absolute accuracy of the clock relative to the PTP epoch. See Table 91 for TimeAccuracy values
	OffsetScaledLogVariance	UINT	Measure of the inherent stability properties of the clock

Attribute ID	Name	Data type	Semantics of values
	CurrentUtcOffset	UINT	Current UTC offset in seconds from International Atomic Time (TAI) of the clock. As of 0 hours 1 January 2006 UTC, the offset was 33 seconds
	TimePropertyFlags	WORD	For the value of the TimePropertyFlags see Table 92
	TimeSource	UINT	For the possible TimeSource values see Table 93
	Priority1	UINT	Relative priority of the grandmaster clock to other clocks in the system. See Priority1 attribute 16
	Priority2	UINT	Relative priority of the grandmaster clock to other clocks in the system. See Priority2 attribute 17
10	LocalClockInfo	STRUCT of	Local Clock information
	ClockIdentity	USINT[8]	Unique identifier of the clock See Table 89 for ClockIdentity encoding
	ClockClass	UINT	Class of the clock quality. Table 90 shows the values most likely to be used by the Time sync object. See IEC 61588:2009 for a complete list of values
	Time Accuracy	UINT	Expected absolute accuracy of the clock relative to the PTP epoch. See Table 91 for TimeAccuracy values
	OffsetScaledLogVariance	UINT	Measure of the inherent stability properties of the clock
	CurrentUtcOffset	UINT	Current UTC offset in seconds from International Atomic Time (TAI) of the clock. As of 0 hours 1 January 2006 UTC, the offset was 33 seconds
	TimeSource	UINT	For the possible TimeSource values see Table 93
	Priority1	UINT	Relative priority of the grandmaster clock to other clocks in the system. See priority1 attribute 16
11	NumberOfPorts	INT	Relative priority of the grandmaster clock to other clocks in the system. See priority2 attribute 17
12	PortStateInfo	STRUCT of	Port state information
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PortState	UINT	The current state of the PTP port: 1 = Initializing 2 = Faulty 3 = Disabled 4 = Listening 5 = Pre_Master 6 = Master 7 = Passive 8 = Uncalibrated 9 = Slave All other = Reserved
13	PortEnableCfg	STRUCT of	Port enable configuration
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PortEnable	UINT	PortEnable has the following values (default is "enabled"): 1 = port enabled 0 = port disabled
14	PortLogAnnounceIntervalCfg	STRUCT of	Port log announce interval configuration

Attribute ID	Name	Data type	Semantics of values
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PortLogAnnounceInterval	UINT	PTP announce interval between successive "Announce" messages issued by a master clock
15	PortSyncLogIntervalCfg	STRUCT of	Port sync log interval configuration
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PortSyncLogInterval	UINT	PTP sync interval between successive "Sync" messages issued by a master clock
16	Priority1	USINT	This attribute specifies the Best Master ranking of this clock and superseeds the clock quality (class, accuracy, and variance)
17	Priority2	USINT	This attribute specifies the Best Master ranking of this clock after clock quality (class, accuracy, and variance) has been evaluated and superseeds the tie breaker
18	DomainNumber	USINT	PTP clock domain
19	ClockType	WORD	For ClockType Values see Table 94
20	ManufactureIdentity	USINT[4]	Manufacturer identity of the clock. The first 3 octets specify the IEEE OUI (Organization Unique Id) for the manufacturer. The last octet is reserved
21	ProductDescription	STRUCT of	
	Size	UDINT	Size of product description
	Description	USINT[Size]	Product description of the device that contains the clock
22	RevisionData	STRUCT of	
	Size	UDINT	Size of revision date
	Revision	USINT[Size]	Revision data of the device that contains the clock
23	UserDescription	STRUCT of	
	Size	UDINT	Size of user description
	Description	USINT[Size]	UserDescription specifies the user description of the device that contains the clock
24	PortProfileIdentityInfo	STRUCT of	Port profile shall be: 00-21-6C-00-01-00
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PortProfileIdentity	USINT[8]	Port profile identity
25	PortPhysicalAddressInfo	STRUCT of	Port physical address information
	NumberOfPorts	UINT	Number of PTP ports on the device
		ARRAY of STRUCT	
	PortNumber	UINT	Number of the port identity
	PhysicalProtocol	UINT	Physical protocol
	SizeOfAddress	UINT	Size of the PortPhysicalAddress
	PortPhysicalAddress	USINT[16]	Port physical address

Attribute ID	Name	Data type	Semantics of values
26	PortProtocolAddressInfo	STRUCT of	Port protocol address information
	NumberOfPorts	UINT	Number of ports
		ARRAY of STRUCT	
	PortNumber	UINT	The number of the port identity
	NetworkProtocol	UINT	Network Protocol values 1 = UDP/IPv4 (e.g. CP 2/2) 2 = UDP/IPv6 3 = ISO/IEC 8802-3 4 = CP 2/3 5 = CP 2/1 0xFFFF = local or unknown protocol all other = reserved
	SizeOfAddress	UINT	Size of the PortProtocolAddress
	PortProtocolAddress	USINT[16]	Port protocol address
27	StepRemoved	UINT	Number of communication paths traversed between the local clock and the grandmaster clock
28	SytemTimeAndOffset	STRUCT of	System time and offset
	SystemTime	ULINT	System time in microseconds
	SystemOffset	ULINT	Offset to the local clock value

Table 89 – ClockIdentity encoding for different network implementations

Network	Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5	Octet 6	Octet 7
CP 2/1	0xFF	0x02	Vendor ID		Serial Number			
CP 2/2	MAC Address			0xFF	0xFE	MAC Address		
CP 2/3	0xFF	0x01	Vendor ID		Serial Number			
Local or closed	0xFF	0xFF	Vendor ID		Serial Number			
All others	See IEC 61588:2009							

Table 90 – ClockClass values

ClockClass	Value
Primary reference	6
Primary reference (Hold)	7
Degraded reference A (Master only)	52
Degraded reference B (Master / Slave)	187
Default	248
Slave only	255
See IEC 61588:2009	all other

Table 91 – TimeAccuracy values

Clock accuracy	Value
Reserved	0x00-0x1F
± 25 ns	0x20
± 100 ns	0x21
± 250 ns	0x22
± 1 µs	0x23
± 2,5 µs	0x24
± 10 µs	0x25
± 25 µs	0x26
± 100 µs	0x27
± 250 µs	0x28
± 1 ms	0x29
± 2,5 ms	0x2A
± 10 ms	0x2B
± 25 ms	0x2C
± 100 ms	0x2D
± 250 ms	0x2E
± 1 s	0x2F
± 10 s	0x30
> ± 10 s	0x31
Reserved	0x32-0x7F
For use by alternate PTP profiles	0x80-0xFD
Unknown	0xFE
Reserved	0xFF

Table 92 – TimePropertyFlags bit values

TimePropertyFlags	Bit index
Leap indicator 61	0
Leap indicator 59	1
Current UTC offset valid	2
PTP timescale	3
Time traceable	4
Frequency traceable	5

Table 93 – TimeSource values

TimeSource	Value
ATOMIC CLOCK	0x10
GPS	0x20
TERRESTRIAL RADIO	0x30
PTP	0x40
NTP	0x50
HAND SET	0x60
OTHER	0x90
INTERNAL OSCILLATOR	0xA0
For use by alternate PTP profiles	0xF0-0xFE
Reserved	0xFF

Table 94 – Types of Clock

Type of Clock	Bit Index
Ordinary clock	0
Boundary clock	1
Peer-to-peer transparent clock	2
End-to-end transparent clock	3
Management node	4
Reserved	All other bits

4.1.8.6.2 Object specific services

There are no object specific services.

4.1.8.7 Parameter object specific syntax elements

4.1.8.7.1 Attributes

4.1.8.7.1.1 Class attributes

The format of the Parameter object class attributes shall be as specified in Table 84.

Table 95 – Parameter object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
2	Max Instance	WORD	The largest instance number of a created object at this class hierarchy level
8	Parameter Class Descriptor	UINT	See Table 96
9	Configuration Assembly Instance	UINT	This attribute shall be set to zero if a configuration assembly is not supported

Attribute ID	Name	Data type	Semantics of values
10	Native Language	UINT	0 = English 1 = French 2 = Spanish 3 = Italian 4 = German 5 = Japanese 6 = Portuguese 7 = Mandarin Chinese

The Parameter Class Descriptor attribute contains bits to describe parameter characteristics. The bits are supported are specified in Table 96.

Table 96 – Parameter Class Descriptor bit values

Bit	Definition
0	Supports Parameter instances 1 = Individual Parameter instances ARE supported. 0 = NO Parameter instances are supported. Only configuration assembly instance used
1	Supports Full attributes 1 = All Full Parameter attributes ARE supported 0 = Only Parameter instance stub attributes are supported
2	Shall do non-volatile storage save command 1 = Shall execute non-volatile storage save command. The Save service of the parameter class is used to save instance parameter values 0 = Need not execute non-volatile storage save command
3	Params are stored in Non-Volatile storage 1 = All Full parameters are stored in non-volatile storage 0 = Parameters are not stored in non-volatile storage

4.1.8.7.1.2 Instance attributes

The format of the Parameter object instance attributes is as specified in Table 85.

Table 97 – Parameter object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	Parameter Value	Data type specified in Descriptor, Data Type and Data Size	
2	Link Path Size	USINT	Number of octets
3	Link Path	Packed EPATH	The Link Path is limited to 255 octets
4	Descriptor	WORD	See Table 98
5	Data Type	EPATH	
6	Data Size	USINT	
7	Parameter Name String	SHORT_STRING	The maximum number of characters is 16
8	Units String	SHORT_STRING	The maximum number of characters is 4
9	Help String	SHORT_STRING	The maximum number of characters is 64
10	Minimum Value	Data Type	See Table 99
11	Maximum Value	Data Type	See Table 99
12	Default Value	Data Type	Actual value the parameter should be set to when the user wants the default for the parameter
13	Scaling Multiplier	UINT	See Table 100
14	Scaling Divisor	UINT	See Table 100
15	Scaling Base	UINT	See Table 100
16	Scaling Offset	INT	See Table 100
17	Multiplier Link	UINT	See Table 101
18	Divisor Link	UINT	See Table 101
19	Base Link	UINT	See Table 101
20	Offset Link	UINT	See Table 101
21	Decimal Precision	USINT	Specifies number of decimal places to use when displaying the scaled engineering value. Also used to determine actual increment value so that incrementing a value causes a change in scaled engineering value to this precision
22	International Parameter Name	STRINGI	Human readable string representing the parameter name
23	International Engineering Units	STRINGI	Engineering units associated with the parameter
24	International Help String	STRINGI	Human readable string representing the help string

Table 98 – Semantics of Descriptor Instance attribute

Bit	Definition	Meaning
0	Supports settable path	Indicates that link path can be set
1	Supports enumerated strings	Indicates that enumerated strings are supported and can be read with the Get_Enum_String service
2	Supports scaling	Indicates that the scaling factor should be implemented to present the value to the user in engineering units

Bit	Definition	Meaning
3	Supports scaling links	Indicates that the values for the scaling factor may be retrieved from other parameters. If the “Supports Scaling” descriptor bit is also set and the scaling link value is zero, the scaling value shall be used. If the “Supports Scaling” descriptor bit is also set and the scaling link value is not zero, the scaling link value shall be used
4	Read only parameter	Indicates that the Parameter Value attribute can only be read, and not set
5	Monitor parameter	Indicates that the Parameter Value attribute is updated in real time by the device
6	Supports extended precision scaling	Indicates that the extended precision scaling factor shall be implemented to present the value to the user in engineering units
7	Supports non-consecutive enumerated strings	Obsolete
8	Allows both enumeration and individual values	Obsolete
9	Non-displayed parameter	The configuration tool shall not display this parameter
10	Indirect parameter reference	The parameter value represents the instance number of another parameter. The referenced parameter instance shall not be an additional indirect reference. A parameter value of 0 shall indicate no reference. The data type for this parameter shall be USINT, UINT or UDINT
11	Not addressable	The parameter is not directly addressable from the network by any of the Set_Attribute or Get_Attribute services
12	Save supported	This bit, when set, indicates that this parameter may be individually saved by sending the Save service to this parameter instance
13	Apply supported	This bit, when set, indicates the execution of the Apply service to this parameter instance is required before attribute changes affect instance behavior
14	Write only parameter	Indicates that the Parameter Value attribute can only be set, and not read
15	Reserved	This shall be set to zero

Table 99 – Minimum and Maximum Value semantics

Data type	Description and semantics	Minimum Value semantics	Maximum Value semantics	Required/Optional/Not allowed
OCTET	Bit String – 8 bit length	The most significant bit that is set in the minimum value is the lowest valid bit to be enumerated. The most significant bit that is set in the maximum value is the highest valid bit to be enumerated. A minimum value of zero means that bit 0 is the lowest valid bit to be enumerated. A maximum value of zero is not valid. Example 1: Min value = 0b00001 Max value = 0b01000 Bits 0 – 3 are enumerated Example 2: Min value = 0b00001 Max value = 0b01011 Bits 0 – 3 are enumerated, bits 0 and 1 in max value are ignored Example 3: Min value = 0b00000 Max value = 0b01000 Bits 0 – 3 are enumerated, 0 min value implies bit 0		Optional
WORD	Bit String – 16 bit length			
DWORD	Bit String – 32 bit length			
LWORD	Bit String – 64 bit length			
STRING ^a	String (2 octet length indicator, 1 octet per character)	Minimum string length	Maximum string length	Required
STRING2 ^a	String (2 octet length indicator, 2 octets per character)	Minimum string length	Maximum string length	Required
STRINGN ^a	String (2 octet length indicator, N octets per character)	Minimum string length	Maximum string length	Required
SHORT_STRING ^a	Character string (1 octet length indicator, 1 octet characters)	Minimum string length	Maximum string length	Required
STRINGI ^a	International character string	Minimum length (in characters) of the "stringcontents" component of the string's implicit sequence	Maximum length (in characters) of the "stringcontents" component of the string's implicit sequence	Required
EPATH ^a	Encoded Path	Minimum string length	Maximum string length	Optional
ENGUNIT	Engineering Units	The minimum value for this data type is not defined and shall not be specified in the EDS file or the parameter object instance	The maximum value for this data type is not defined and shall not be specified in the EDS file or the parameter object instance	Optional
All other data types		The minimum numeric value that may be assigned to the data value	The maximum numeric value that may be assigned to the data value	Optional ^b
^a The STRING, STRING2, STRINGN, SHORT_STRING, STRINGI and EPATH data types do not have a minimum or maximum value specification. The minimum and maximum value fields are used to present the minimum and maximum string or path lengths. In these cases, the Data Size parameter is used to represent the number of octets required per character or encoding entry. ^b If the Minimum Value and/or Maximum Value is not specified then the minimum and/or maximum value for the parameter data value are as defined in IEC 61158-5-2:2010.				

Table 100 – Scaling Formula attributes

Attribute	Meaning
Multiplier Value	(Mult) Multiplier value for the scaling formula. This value can be based on another parameter
Divisor Value	(Div) Divisor value for the scaling formula. This value can be based on another parameter specified in the Divisor Link
Base Value	(Base) Base value for the scaling formula. This value can be based on another parameter specified in the Base Link
Offset Value	(Offset) Offset value for the scaling formula. This value can be based on another parameter specified in the Offset Link. This attribute is an INT and can be negative
Decimal Precision	(Precision) Precision value for the scaling formula. This value cannot be based on another parameter

Scaling Links

Scaling values (multiplier, divisor, base, and offset) can also be based on other parameters. Scaling Links specify from which instance that scaling value is to be retrieved. If the scaling link attribute is set to 0, then that scaling value is a constant and not linked to another parameter. Table 101 specifies the scaling links.

Table 101 – Scaling links

Attribute	Meaning
Multiplier Link	Specifies the parameter instance from where the Multiplier Value is retrieved
Divisor Link	Specifies the parameter instance from where the Divisor Value is retrieved
Base Link	Specifies the parameter instance from where the Base Value is retrieved
Offset Link	Specifies the parameter instance from where the Offset Value is retrieved

Scaling Factor Attributes

The Scaling Factor represents actual UINT and INT parameter values in other formats. Formula (1) shall be used to determine the engineering value from the actual value.

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) \times \text{Mult} \times \text{Base}}{\text{Div}} \quad (1)$$

The engineering value can then be displayed to the user in the terms specified within the Decimal Precision Instance Attribute. The inverse of the Scaling Formula shall be used to determine the actual value from the engineering value (see Formula (2)).

$$\text{ActualValue} = \frac{(\text{EngValue} \times \text{Div})}{\text{Mult} \times \text{Base}} - \text{Offset} \quad (2)$$

The extended precision scaling factor adds extended precision to the standard scaling factor. Formula (3) shall be used to determine the engineering value from the actual value.

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) \times \text{Mult} \times \text{Base}}{\text{Div} \times 10^{\text{Precision}}} \quad (3)$$

The engineering value can then be displayed to the user in the terms specified within the Decimal Precision attribute. The inverse of the extended precision scaling formula shall be used to determine the actual value from the engineering value (see Formula (4)).

$$\text{ActualValue} = \frac{(\text{EngValue} \times \text{Div} \times 10^{\text{Precision}})}{\text{Mult} \times \text{Base}} - \text{Offset} \quad (4)$$

4.1.8.7.2 Object specific service

Get_Enum_String

The Get_Enum_String_RequestPDU body shall be as specified in Table 102.

Table 102 – Structure of Get_Enum_String_RequestPDU body

Name	Data type	Semantics of values
Enumerated String Number	USINT	Number of enumerated strings to retrieve. Maximum possible range of values = 0 to 255. Value indicates bit offset (relative to zero) for bit enumerations, actual value (relative to zero) for value enumerations

The Get_Enum_String_ResponsePDU body shall be as specified in Table 103.

Table 103 – Structure of Get_Enum_String_ResponsePDU body

Name	Data type	Semantics of values
Enumerated String	SHORT_STRING	Enumerated strings. Maximum number of characters = 16

Enumerated strings are human-readable strings that describe either a bit or a value, depending on the parameter's data type. The relationship of the string type with the parameter data type is shown in Table 104.

Table 104 – Enumerated strings Type versus Parameter data type

If the parameter's data type is:	Then the enumerated strings returned are:
OCTET, WORD, DWORD, LWORD	Bit enumerated strings
USINT, SINT, UINT, INT, BOOL	Value enumerated strings
All Other data types	Invalid for enumeration

If the parameter's data type is OCTET, WORD, DWORD or LWORD, requesting a Get_Enum_String service with an enumerated string number of 0 on a parameter returns a SHORT_STRING that describes bit 0. Requesting a Get_Enum_String service with an enumerated string number of 1 on a parameter returns a SHORT_STRING that describes bit 1. This continues up to the maximum bit number supported by the parameter.

If the parameter's data type is SINT, USINT, INT, UINT or BOOL, requesting a Get_Enum_String service with an enumerated string number of 0 on a parameter returns a SHORT_STRING that describes the value of 0. Requesting a Get_Enum_String service with an enumerated string number of 1 returns a SHORT_STRING that describes the value of 1.

4.1.8.8 Connection Manager object specific syntax elements

4.1.8.8.1 Attributes

4.1.8.8.1.1 Class attributes

The format of the Connection Manager object class attributes shall be as specified in Table 105.

Table 105 – Connection Manager object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
2	Max Instance	UDINT	
4	Optional Attribute List	STRUCT of	
	Number attributes	UINT	
	Optional attributes	ARRAY of UINT	

4.1.8.8.1.2 Instance attributes

The format of the Connection Manager object instance attributes is as specified in Table 106

Table 106 – Connection Manager object instance attributes

Attribute ID	Name	Data type	Semantics of values
1	OpenReqs	UINT	
2	OpenFormat Rejects	UINT	
3	OpenResource Rejects	UINT	
4	OpenOther Rejects	UINT	
5	CloseReqs	UINT	
6	CloseFormat Rejects	UINT	
7	CloseOther Rejects	UINT	
8	ConnTimeouts	UINT	Total number of connection timeouts that have occurred in connections controlled by this Connection Manager.
9	Connection Entry List	STRUCT of	
	NumConnEntries	UINT	Number of bits used in the ConnOpenBits element. This attribute, divided by 8 and incremented for any remainder, gives the length in octets of the array in the ConnOpenBits field.
	ConnOpenBits	ARRAY of BOOL	Bit field. List of connection data. Each bit represents a possible connection. 0 = No Connection. 1 = Connection Established.
11	CpuUtilization	UINT	CPU Utilization in tenths of a percent. Range of 0 – 1 000 representing 0 to 100%.
12	MaxBuffSize	UDINT	Amount of buffer space is in octets.
13	BufSize Remaining	UDINT	Amount of buffer space is in octets.

4.1.8.9 Connection object specific syntax elements

4.1.8.9.1 Attributes

4.1.8.9.1.1 Class attributes

The format of the Connection object class attributes shall be as specified in Table 107.

Table 107 – Connection object class attributes

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	Revision of this object. Value = 1
2	Max Instance	UINT	

4.1.8.9.1.2 Instance attributes

The format of the Connection object instance attributes is as specified in Table 108.

Table 108 – Connection object instance attributes

Attr ID	Name	Data Type	Semantics
1	State	USINT	State of the object.
2	Instance_type	USINT	Indicates either I/O or Messaging Connection.
3	TransportClass_trigger	SWORD	Defines behavior of the Connection.
4	CP2/3_produced_connection_id	UINT	Placed in ISO 11898 CAN Identifier Field when the Connection transmits on a CP 2/3 subnet.
5	CP2/3_consumed_connection_id	UINT	ISO 11898CAN Identifier Field value that denotes message to be received on a CP 2/3 subnet.
6	CP2/3_initial_comm_characteristics	SWORD	Defines the Message Group(s) across which productions and consumptions associated with this Connection occur on a CP 2/3 subnet.
7	Produced_connection_size	UINT	Maximum number of octets transmitted across this Connection.
8	Consumed_connection_size	UINT	Maximum number of octets received across this Connection.
9	Expected_packet_rate	UINT	Defines timing associated with this Connection.
10	CFP2_produced_connection_id	UDINT	Identifies the message sent on the subnet by this connection.
11	CPF2_consumed_connection_id	UDINT	Identifies the message received from the subnet for this connection.
12	Watchdog_timeout_action	USINT	Defines how to handle Inactivity/Watchdog timeouts.
13	Produced_connection_path_length	UINT	Number of octets in the produced_connection_path attribute.
14	Produced_connection_path	Packed EPATH	Specifies the application object(s) whose data is to be produced by this Connection object. See 4.1.9.
15	Consumed_connection_path_length	UINT	Number of octets in the consumed_connection_path attribute.
16	Consumed_connection_path	Packed EPATH	Specifies the application object(s) that are to receive the data consumed by this Connection object. See 4.1.9.
17	Production_inhibit_time	UINT	Defines minimum time between new data production. This attribute is required for all I/O Client connections, except those with a production trigger of Cyclic.
18	Connection_timeout_multiplier	USINT	Specifies the multiplier applied to the expected_packet_rate value to derive the value for the Inactivity/Watchdog Timer.
19	Connection_binding_list	STRUCT UINT Array of UINT	List of I/O connection instances bound to this instance.

State

This attribute defines the current state of the Connection instance. Table 109 defines the possible states and assigns a value used to indicate that state.

Table 109 – Values assigned to the state attribute

Value	State Name	Description
00	Non-existent	The Connection has yet to be instantiated.
01	Configuring	The Connection has been instantiated and is waiting for the following events to occur: (1) to be properly configured and (2) to be told to apply the configuration.
02	Waiting For Connection ID ^a	The Connection instance is waiting exclusively for its consumed_connection_id and/or produced_connection_id attribute to be set. ^b
03	Established	The Connection has been validly/fully configured and the configuration has been successfully applied.
04	Timed Out	If a Connection object experiences an Inactivity/Watchdog timeout, then a transition <u>may</u> be made to this state. See the watchdog_timeout_action attribute description and the description of the Inactivity/Watchdog for more details.
05	Deferred Delete ^a	If an Explicit Messaging Connection object experiences an Inactivity/Watchdog timeout, then a transition <u>may</u> be made to this state. See the watchdog_timeout_action attribute description and the description of the Inactivity/Watchdog Timer for more details.
06	Closing	A Bridged Connection object has received, and is processing, a Forward Close from the Connection Manager. The deletion of the connection does not occur until after a successful Forward Open response has been received from the target node.
^a This value is only used on CP 2/3.		
^b When the Connection instance attributes are applied it may not be possible for the module to generate the produced_connection_id and/or consumed_connection_id values (see initial_comm_characteristics attribute for rules). If this is the case then all the tasks required to apply the attributes are performed except for the initialization of these attributes within the Connection and associated Link Producer/Consumer objects and a transition is made to this state. In this state the Connection instance is waiting exclusively for its produced_connection_id and/or consumed_connection_id attributes to be set.		

Important: A dynamically created connection instance is the child of the Explicit Messaging connection across which it was created. Different subnet types may provide other mechanisms for creating a connection that imply a particular parent-child relationship. See network-specific specifications for details.

Important: All resources associated with a Connection Instance (A) that has been dynamically created across an Explicit Messaging Connection (B) shall be released if the Explicit Messaging Connection (B) times out prior to the dynamically created Connection Instance (A) transitioning to the Established state.

Important: When a transition is made to the Established state, all timers associated with the Connection object are activated (see Connection Timing in IEC 61158-5-2:2010, 6.2.3.2.1.7)

Instance_type

This attribute defines the instance type (see Table 110).

Table 110 – Values assigned to the instance_type attribute

Value	Meaning
00	Explicit Messaging. This Connection Instance represents one of the end-points of an Explicit Messaging Connection. An Explicit Messaging Connection is dynamically created by sending the <i>Open Explicit Messaging Connection Request</i> to the Connection Class.
01	I/O. This Connection Instance represents one of the end-points of an I/O Connection. An I/O Connection is dynamically created by sending a <i>Create Request</i> to the Connection Class.
02	Bridged. This Connection Instance represents an intermediate 'hop' of a bridged I/O or Explicit Messaging connection. A pair of Bridged Connection objects (one for each subnet bridged between) are dynamically created by a node after successfully receiving a Forward Open service to the Connection Manager object when this node is not the end point (target).

TransportClass_trigger

Defines whether this is a producing only, consuming only, or both producing and consuming connection. If this end point is to perform a data production, this attribute also defines the event that triggers the production. The eight (8) bits are divided as shown in Figure 7.

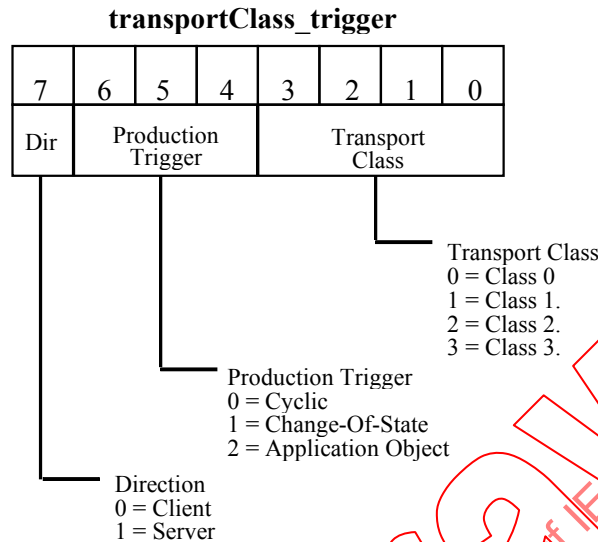


Figure 7 – Transport Class Trigger attribute

The **Direction bit** of the transportClass_trigger octet indicates whether the end-point is to act as the Client or the Server on this connection (see Table 111).

Table 111 – Possible values within Direction Bit

Value	Meaning	
0	Client	This end-point provides the Client behavior associated with this Connection. Additionally, this value indicates that the <i>Production Trigger</i> bits within the transportClass_trigger octet contain the description of <u>when</u> the Client is to produce the message associated with this connection. Client connections with production trigger value of 0 or 1 (Cyclic or Change-of-State) shall produce immediately after transitioning to the Established state.
1	Server	This end-point provides the Server behavior associated with this Connection. In addition, this value indicates that the <i>Production Trigger</i> bits within the transportClass_trigger octet are to be IGNORED. The <i>Production Trigger</i> bits are ignored due to the fact that a Server end-point <i>reacts</i> to the transmission from the Client. The only means by which a Server end-point is triggered to transmit is when this <i>reaction</i> calls for the production of a message (Transport Classes 2 or 3).

Table 112 lists the values that are possible within the **Production Trigger** bits of the transportClass_trigger attribute.

Table 112 – Possible values within Production Trigger Bits

If the value is:	Then the Production of a message is:	
0	Cyclic	The expiration of the Transmission Trigger Timer triggers the data production. See IEC 61158-5-2:2010, 6.2.3.2.1.7 Connection Timing for a detailed description of the Transmission Trigger Timer.

If the value is:	Then the Production of a message is:	
1	Change-Of-State	Production occurs when a change-of-state is detected by the application object. NOTE The consuming end-point may have been configured to expect the packet at a certain rate, regardless of the triggering mechanism at the producing end-point. See the description of the <code>expected_packet_rate</code> attribute of a Connection object and the description of Connection Timing in IEC 61158-5-2:2010, 6.2.3.2.1.7 for more information.
2	Application Object Triggered	The application object decides when to trigger the production. NOTE The consuming end-point may have been configured to expect the packet at a certain rate, regardless of the triggering mechanism at the producing end-point. See the description of the <code>expected_packet_rate</code> attribute of a Connection object and the description of Connection Timing in IEC 61158-5-2:2010, 6.2.3.2.1.7 for more information.
3 - 7	Reserved	

Table 113 lists possible values within the **Transport Class** nibble of the **transportClass_trigger** attribute. Behaviors resulting from these particular values are illustrated in the series of figures that follow the table.

Table 113 – Possible values within Transport Class Bits

Value	Meaning	
0	Transport Class 0	Based on the value within the <i>Dir</i> bit, this connection end-point will be a producing only OR consuming only end-point. Upon application of this Connection instance, the module instantiates either a Link Producer (<i>Dir</i> bit = Client, producing only) or a Link Consumer (<i>Dir</i> bit = Server, consuming only) to be associated with this Connection.
1	Transport Class 1	
2	Transport Class 2	Indicates that the module will both produce AND consume across this connection. The Client end-point generates the first data production that is consumed by the Server, which causes the Server to return a production that is consumed by the Client.
3	Transport Class 3	
4	Transport Class 4	Non-blocking.
5	Transport Class 5	Non-blocking, fragmenting.
6	Transport Class 6	Multicast, fragmenting.
7 - F	Reserved	

A 16-bit sequence count value is prepended to all Class 1, 2, and 3 transports. This value is used to detect delivery of duplicate data packets. Sequence count values are initialized on the first message production and incremented on each subsequent new data production. A resend of old data shall not cause the sequence count to change and a consumer shall ignore data when it is received with a duplicate sequence count. Consuming applications can use this mechanism to distinguish between new samples and old samples that were sent to maintain the connection.

However, on CP 2/3, transport classes 2 and 3 do not prepend a 16-bit sequence count as described in 4.1.4.

The following tables and figures illustrate the valid combinations of **Production Trigger** and **Transport Class** and provides a description of the Client and Server behaviors. See IEC 61158-5-2:2010, 6.2.3.2.1.7 for a description of the **Transmission Trigger Timer**, which is shown in the illustrations.

To summarize, refer to Table 114 for the valid values within the **transportClass_trigger** attribute of a Connection Instance:

Table 114 – Transport Class_Trigger attribute

TransportClass_trigger bits	Meaning
1 xxx 0000	Direction = Server, Production Trigger = IGNORED, Transport Class = 0.
1 xxx 0001	Direction = Server, Production Trigger = IGNORED, Transport Class = 1.
1 xxx 0010	Direction = Server, Production Trigger = IGNORED, Transport Class = 2.
1 xxx 0011	Direction = Server, Production Trigger = IGNORED, Transport Class = 3. This is the value assigned to this attribute within the Server end-point of an Explicit Messaging Connection.
0 000 0000	Direction = Client, Production Trigger = Cyclic, Transport Class = 0.
0 000 0001	Direction = Client, Production Trigger = Cyclic, Transport Class = 1.
0 000 0010	Direction = Client, Production Trigger = Cyclic, Transport Class = 2.
0 000 0011	Direction = Client, Production Trigger = Cyclic, Transport Class = 3.
0 001 0000	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 0.
0 001 0001	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 1.
0 001 0010	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 2.
0 001 0011	Direction = Client, Production Trigger = Change-Of-State, Transport Class = 3.
0 010 0000	Direction = Client, Production Trigger = Application object, Transport Class = 0.
0 010 00001	Direction = Client, Production Trigger = Application object, Transport Class = 1.
0 010 0010	Direction = Client, Production Trigger = Application object, Transport Class = 2.
0 010 0011	Direction = Client, Production Trigger = Application object, Transport Class = 3. This is the value assigned to this attribute within the Client end-point of an Explicit Messaging Connection.
1 111 1111	Default value assigned to this attribute within an I/O Connection.

CP2/3_produced_connection_id

This attribute is Required for a CP 2/3 subnet. Other subnet types shall not use this attribute.

It contains the CP 2/3 Connection ID to be associated with transmissions sent across this connection (if any), i.e. the value that will be specified in the CAN Identifier Field of ISO 11898 when this Connection transmits. See IEC 62026-3:2008, 5.1.2 for a description of how CP 2/3 uses the CAN Identifier Field. This value is loaded directly into the associated Link Producer's connection_id attribute. Values are defined in Table 115.

Table 115 – Values defined for the CP2/3_produced_connection_id attribute

Value	Meaning
0 - 7F _{hex}	The value to be placed in the CAN Identifier Field when this Connection transmits.
800 _{hex} - FFFE _{hex}	Reserved
FFFF _{hex}	Default value assigned to this attribute within an I/O Connection. This attribute will retain this value if this Connection instance is not producing any data (consumer only).

CP2/3_consumed_connection_id

This attribute is Required for a CP 2/3 subnet. Other subnet types shall not use this attribute.

It contains the Connection ID which identifies messages to be received across this connection (if any), i.e. the CAN Identifier Field value that is associated with messages this Connection object receives. See IEC 62026-3:2008, 5.1.2 for a description of how CP 2/3 uses the CAN Identifier Field. This value is loaded directly into the associated Link Consumer's connection_id attribute. Values are defined in Table 116.

Table 116 – Values defined for the CP2/3_consumed_connection_id attribute

Value	Meaning
0 - 7F _{hex}	The value that identifies messages to be consumed. This will be specified in the CAN Identifier Field of messages that are to be consumed.
800 _{hex} - FFFE _{hex}	Reserved
FFFF _{hex}	Default value assigned to this attribute within an I/O Connection. This attribute will retain this value if this Connection instance is not consuming any data (producer only).

CP2/3_initial_comm_characteristics

This attribute is Required for a CP 2/3 subnet. Other subnet types shall not use this attribute.

It defines the Message Group(s) across which productions and consumptions associated with this Connection occur.

This octet is divided into two nibbles, as show in Figure 8.

7	6	5	4	3	2	1	0
Initial Production Characteristics				Initial Consumption Characteristics			

Figure 8 – CP2/3_initial_comm_characteristics attribute format

Table 117 lists the values that are possible within the Initial Production Characteristics nibble (upper nibble) of the CP2/3_initial_comm_characteristics attribute.

Table 117 – Values for the Initial Production Characteristics nibble

Value	Meaning	
0	Produce across Message Group 1	The production associated with this Connection is to take place across Message Group 1. The producing module generates the Connection ID value and loads it into the Connection object's CP2/3_produced_connection_id attribute. The producing module allocates a Message ID from its Group 1 Message ID pool and combines this with its Source MAC ID to generate the Connection ID. The numerically lowest available Group 1 Message ID is to be used in generating the CP2/3_produced_connection_id attribute value. This value shall also be loaded into the corresponding CP2/3_consumed_connection_id attribute(s) associated with the consuming Connection object(s).
1	Produce across Message Group 2 (Destination)	The production associated with this Connection is to take place across Message Group 2. Additionally, the intended recipient's MAC ID (Destination MAC ID) is to be placed within the MAC ID component of the Group 2 Identifier Field. In this case, the consuming module generates the Connection ID value to be associated with transmissions across this connection. When the consuming module has generated this value and loaded it into the appropriate Connection object's CP2/3_consumed_connection_id attribute, it can be read and subsequently loaded into the producing Connection object's CP2/3_produced_connection_id attribute.
2	Produce across Message Group 2 (Source)	The production associated with this Connection is to take place across Message Group 2. In addition, the producing module's MAC ID (Source MAC ID) is to be placed within the MAC ID component of the Group 2 Identifier. In this case, the producing module generates the Connection ID value and loads it into the Connection object's CP2/3_produced_connection_id attribute. The numerically lowest available Group 2 Message ID is to be used in generating the CP2/3_produced_connection_id attribute value. This value shall also be loaded into the corresponding CP2/3_consumed_connection_id attribute(s) associated with the consuming Connection object(s).
3	Produce across Message Group 3	The production associated with this Connection is to take place across Message Group 3. The producing module generates the Connection ID value and loads it into the Connection object's CP2/3_produced_connection_id attribute. The producing module allocates a Message ID from its Group 3 Message ID pool and combines this with its Source MAC ID to generate the Connection ID. The numerically lowest available Group 3 Message ID is to be used in generating the CP2/3_produced_connection_id attribute value. This value shall also be loaded into the corresponding CP2/3_consumed_connection_id attribute(s) associated with the consuming Connection object(s).
4 - 0xE	Reserved	
0xF	Default value	The default value assigned to the CP2/3 Initial Production Characteristics nibble within an I/O Connection. NOTE If this is a consuming only I/O Connection, then the default value remains in this nibble. Explicit Messaging Connection objects automatically configure this attribute when the Connection is established.

Table 118 lists the possible values within the Initial Consumption Characteristics nibble (lower nibble) of the CP2/3_initial_comm_characteristics attribute.

Table 118 – Values for the Initial Consumption Characteristics nibble

Value	Meaning	
0	Consume a Group 1 Message	The message to be consumed will be transmitted across Message Group 1. The producing module generates the Connection ID value. This value shall be loaded into the CP2/3_consumed_connection_id attribute associated with the consuming Connection object(s).
1	Consume a Group 2 Message (Destination)	The message to be consumed will be transmitted across Message Group 2. The intended recipient's MAC ID (Destination MAC ID) is specified within the Group 2 Identifier. The consuming module generates the Connection ID value and loads it into the CP2/3_consumed_connection_id attribute associated with this Connection object. The numerically lowest available Group 2 Message ID is to be used in generating the CP2/3_consumed_connection_id attribute value. This value shall be loaded into the producing Connection object's CP2/3_produced_connection_id attribute.
2	Consume a Group 2 Message (Source)	The message to be consumed will be transmitted across Message Group 2. The transmitting module's MAC ID (Source MAC ID) is specified within the Group 2 Identifier. In this case, the producing module generates the Connection ID value and loads it into the Connection object's the CP2/3_produced_connection_id attribute. This value shall be loaded into the CP2/3_consumed_connection_id attribute associated with the consuming Connection object(s).
3	Consume a Group 3 Message	The message to be consumed will be transmitted as a Group 3 Message. The producing module generates the Connection ID value. The Connection ID value shall be loaded into this Connection object's CP2/3_consumed_connection_id attribute.
4 - 0xE	Reserved	
0xF	Default value	The default value assigned to the CP 2/3 Initial Consumption Characteristics nibble within an I/O Connection. NOTE If this is a producing only I/O Connection, then the default value remains in this nibble. Explicit Messaging Connections automatically configure this attribute when the Connection is established.

Important: The module that generates a Connection ID shall guarantee that it does not allocate the Message ID/MAC ID pair in such a way that two separate modules are capable of transmitting identical bit patterns within the Identifier Field.

Produced_connection_size

The meaning of this attribute is different for Explicit Messaging Connections than it is for I/O Connections. If the subnet defines a fragmentation protocol and the device supports fragmentation, this size may be larger than the largest frame size. See the network specific adaptation specifications for more details.

For Explicit Messaging Connections:

This attribute signifies the maximum number of Message Router Request/Response Data octets (see 4.1.8) that a device is able to transmit across this Connection. Devices that place a known limit on the maximum amount of Message Router Request/Response Data that can be transmitted in a single message, or single fragmented series initialize this attribute accordingly. Devices that cannot or do not predefine an up-front transmit limit place the value 0xffff into this attribute (there may still be a limit, however, it is not known in advance).

Important: Due to the nature of Explicit Messaging, the length of Explicit Messages will fluctuate over the lifetime of a connection. Explicit Messaging Connections perform fragmentation based on the *length* of the current message to transmit where the subnet type may define a fragmentation protocol.

For I/O Connections:

If the **transportClass_trigger** indicates that this Connection instance is to produce, then this attribute defines the maximum amount of I/O data that may be produced as a *single unit* across this connection. The amount of I/O to be transmitted at any given point in time can be less than or equal to the **connection_size** attribute.

This attribute defaults to zero (0) within an I/O Connection. If the subnet type supports fragmentation and this attribute is set to a value greater than the largest payload in an I/O Connection, then the Connection will break up the data into multiple fragments. See the network specific adaptation specifications for more details.

Important: Fragmentation within I/O Connections is performed based on the value within this attribute, regardless of the current amount of data to transmit.

Important: I/O Messages that contain no Application I/O Data and were configured to contain data (via produced_connection_size being greater than zero (0)) are defined to indicate a *No Data* event for the receiving application object(s). The behavior of an application object upon detection of the *No Data* event is application object specific.

Consumed_connection_size

The meaning of this attribute is different for Explicit Messaging Connections than it is for I/O Connections. If the subnet defines a fragmentation protocol and the device supports fragmentation, this size may be larger than the largest frame size. See the network specific adaptation specifications for more details.

For Explicit Messaging Connections:

This attribute signifies the maximum number of Message Router Request/Response Data octets that a device is able to receive across this Connection.

Devices that place a known limit on the maximum amount of Message Router Request/Response Data that can be received in a single message, or single fragmented series initialize this attribute accordingly. Devices that cannot or do not predefine an up-front receive limit place the value 0xffff into this attribute (there may still be a limit, however, it is not known in advance). Because of the nature of Explicit Messaging, the length of Explicit Messages will fluctuate over the lifetime of a connection.

For I/O Connections:

If the **transportClass_trigger** attribute indicates that this Connection is to consume, then this attribute defines the maximum amount of data that may be received as a *single unit* across this connection. The actual amount of I/O data received at any given time can be less than or equal to the **connection_size** attribute.

This attribute defaults to zero (0) within an I/O Connection. If the subnet type supports fragmentation and this attribute is set to a value greater than the largest payload in an I/O Connection, then the Connection will process the fragmentation protocol. See the network specific adaptation specifications for more details.

The length of an I/O Message shall be less than or equal to this attribute for an I/O Connection object to receive it as a valid message. If an I/O Connection object receives a message whose length is greater than this attribute, then it immediately discards the message and discontinues any subsequent processing.

NOTE With respect to the Server end-point of a Transport Class 2 or 3 Connection, this *too much data* error condition results in no response being transmitted and the watchdog timer is not kicked.

Expected_packet_rate

This attribute is used to generate the values loaded into the *Transmission Trigger Timer* and the *Inactivity/Watchdog Timer*. See IEC 61158-5-2:2010, 6.2.3.2.1.7 for a description of the Transmission Trigger and Inactivity/Watchdog timers.

The resolution of this attribute is in milliseconds. A request to configure this attribute may result in the specification of a time value that a product cannot meet. In addition to performing product specific range checking when a request to modify this attribute is received, the following steps are performed:

- If the specified value is not equal to an increment of the available clock resolution, then the value is rounded **up** to the next serviceable value. For example: a Set_Attribute_Single request is received specifying the value 5 for the **expected_packet_rate** attribute and the product provides a 10 millisecond resolution on timers. In this case the product would load the value 10 into the **expected_packet_rate** attribute.
- The value that is actually loaded into the **expected_packet_rate** attribute is reported in the Service Data Field of a Set_Attribute_Single response message associated with a request to modify this attribute.
- If the requested value is equal to an increment of the clock resolution, then the requested value is loaded into the **expected_packet_rate** and reported in the response. For example: if the value 100 is requested and the clock resolution is 10 milliseconds, then of a value of 100 is loaded.
- When a Connection object is in the **Established** state, any modifications to the **expected_packet_rate** attribute have immediate effect on the Inactivity/Watchdog Timer. The following steps are performed by a Connection object in the **Established** state when a request is received to modify the **expected_packet_rate** attribute:
 - the current Inactivity/Watchdog Timer is canceled,
 - a new Inactivity/Watchdog Timer is activated based on the new value in the **expected_packet_rate** attribute.

This attribute defaults to 2500 (2 500 ms) within Explicit Messaging Connections, and to zero (0) within an I/O Connection.

CPF2_produced_connection_id

Contains the Connection ID, which identifies messages to be sent across this connection (if any). This attribute shall not be implemented when the subnet type is CP 2/3.

CPF2_consumed_connection_id

Contains the Connection ID, which identifies messages to be received across this connection (if any). This attribute shall not be implemented when the subnet type is CP 2/3.

Watchdog_timeout_action

This attribute defines the action the Connection object should perform when the Inactivity/Watchdog Timer expires. Table 119 defines the specifics of this attribute.

Table 119 – Values for the watchdog_timeout_action

Value	Meaning
0	<i>Transition to Timed Out.</i> The Connection transitions to the Timed Out state and remains in this state until it is Reset or Deleted. The command to Reset or Delete could come from an internal source (e.g., an application object) or could come from the network (e.g., a configuration tool). This is the default value for this attribute with respect to I/O Connections. This value is invalid for Explicit Messaging Connections.
1	<i>Auto Delete.</i> The Connection Class automatically deletes the Connection if it experiences an Inactivity/Watchdog timeout. This is the default value for this attribute with respect to Explicit Messaging Connections.
2	<i>Auto Reset.</i> The Connection remains in the Established state and immediately restarts the Inactivity/Watchdog timer. This value is invalid for Explicit Messaging Connections.
3	<i>Deferred Delete.</i> The Connection transitions to the Deferred state if any child connection instances are in the Established state. If no child connection instances are in the Established state the connection is deleted. This value is only used on CP 2/3 and is invalid for I/O Messaging Connections.
4 - FF	Reserved

Produced_connection_path_length

Specifies the number of octets of information within the **produced_connection_path** attribute. This is automatically initialized when the **produced_connection_path** attribute is configured. This attribute defaults to the value zero (0).

Produced_connection_path

The **produced_connection_path** attribute is made up of a octet stream which defines the application object(s) whose *data* is to be produced by this Connection object. **The format of this octet stream is specified in 4.1.9.** This attribute defaults to being empty upon instantiation of the Connection. It remains empty within Explicit Messaging Connections and within Connection objects that do not produce.

Consumed_connection_path_length

Specifies the number of octets of information within the **consumed_connection_path** attribute. This is automatically initialized when the **consumed_connection_path** attribute is configured. This attribute defaults to the value zero (0).

Consumed_connection_path

The **consumed_connection_path** attribute is made up of an octet stream which defines the application object(s) that are to receive the *data* consumed by this Connection object. **The format of this octet stream is specified in 4.1.9.** This attribute defaults to being empty upon instantiation of the Connection. It remains empty within Explicit Messaging Connections and within Connection objects that do not consume.

Production_inhibit_time

This attribute is used to configure the minimum delay time between new data production. This is required for all I/O Client connections, except those with a production trigger of Cyclic. The Set_Attribute_Single service shall be supported when this attribute is implemented. A value of zero (the default value for this attribute) indicates no inhibit time.

The resolution of this attribute is in milliseconds. A request to configure this attribute may result in the specification of time value that a product cannot meet. In addition to performing product specific range checking when a request to modify this attribute is received, the following steps are performed:

- If the specified value is not equal to an increment of the available clock resolution, then the value is rounded **up** to the next serviceable value. For example: a Set_Attribute_Single request is received specifying the value 5 for the **production_inhibit_time** attribute and the product provides a 10 millisecond resolution on times. In this case the product would load the value 10 into the **production_inhibit_time** attribute.

- The value that is actually loaded into the **production_inhibit_time** attribute is reported in the Service Data Field of a Set_Attribute_Single response message associated with a request to modify this attribute.
- If the requested value is equal to an increment of the clock resolution, then the requested value is loaded into the **production_inhibit_time** and reported in the response. For example: the value 100 is requested and the clock resolution is 10 milliseconds.

The **production_inhibit_time** value is loaded in the Production Inhibit Timer each time new data production occurs.

When a Connection object is in the **Established** state, any modifications to the **production_inhibit_time** attribute have no effect on a currently running Production Inhibit Timer. The new production_inhibit_time value is loaded into the Production Inhibit Timer on the following new data production.

When the apply_attributes service is received, the **production_inhibit_time** shall be verified against the **expected_packet_rate** attribute. If the **expected_packet_rate** value is greater than zero, but less than the **production_inhibit_time** value, then an error shall be returned. In this case, where two attribute values conflict use the **production_inhibit_time attribute ID** as the additional error code returned in the error response.

Connection_timeout_multiplier

The Connection_timeout_multiplier specifies the multiplier applied to the expected packet rate to obtain the value used by the Inactivity/Watchdog Timer. See the Connection Timeout Multiplier parameter description within the Connection Manager object Specific Service Parameters for the enumerated values of this attribute. The default value for this attribute is zero (specifying a multiplier of 4).

Connection_binding_list

The Connection_binding_list attribute identifies connection instances that are bound to this connection. The attribute structure provides a 16 bit count of bound connections, followed by a list of the bound instances. This attribute shall be supported if the Connection_Bind service is supported.

4.1.8.9.2 Object specific services

4.1.8.9.2.1 Connection_Bind Service

The Connection_Bind object specific service binds two dynamically created I/O connection instances together for purposes of connection timeouts and deletions. A dynamically created I/O connection is the result of a Create service to the Connection class with the Instance_type attribute set to I/O (attribute value of 1). If one of these I/O connection instance times out or is deleted, the other instance shall exhibit the same behavior.

The shall be as specified in Table 120.

Table 120 – Structure of Connection_Bind_RequestPDU body

Parameter Name	Data Type	Parameter Description
Bound Instances	STRUCT of UINT UINT	Instance numbers of Connection objects to be bound.

The Connection_Bind_ResponsePDU body is empty.

The service response may return a status from the list of status codes in Table 121.

Table 121 – Object specific status for Connection_Bind service

General Status Code	Extended Status Code	Status Description
0x02	0x01	One or both of the connection instances is non-existent.
0x02	0x02	The connection class and/or instance is out of resources to bind instances.
0x0C	0x01	Both of the connection instances are existent, but at least one is not in the Established state.
0x20	0x01	Both connection instances are the same value.
0xD0	0x01	One or both of the connection instances is not a dynamically created I/O connection.
0xD0	0x02	One or both of the connection instances were created internally and the device is not allowing a binding to it.

4.1.8.9.2.2 Producing_Application_Lookup Service

The Producing_Application_Lookup object specific service provides a mechanism to find one or more connection instances in the Established state producing data from a given application object. If the requested producing application path is being produced by any connection in the Established state, the instance number of each connection producing from that path is returned.

The Producing_Application_Lookup_RequestPDU body shall be as specified in Table 122.

Table 122 – Structure of Producing_Application_Lookup_RequestPDU body

Parameter Name	Data Type	Parameter Description
Producing Application Path	EPATH	Connection path of producing application to be searched for within connections in the Established state. The path shall be a single Logical or Symbolic EPATH.

The Producing_Application_Lookup_ResponsePDU body shall be as specified in Table 123.

Table 123 – Structure of Producing_Application_Lookup_ResponsePDU body

Parameter Name	Data Type	Parameter Description
Instance Count	UINT	Number of Instances returned in the Connection Instance List parameter.
Connection Instance List	Struct of UINT	List of instance numbers of connection producing the data from the requested connection path within the node.

The service response may return a status from the list of status codes in Table 124.

Table 124 – Producing_Application_Lookup Service status codes

General Status Code	Extended Status Code	Status Description
0x02	0x01	The connection path was not found in any connection instance in the Established state.

4.1.8.9.2.3 SafetyOpen Service

See IEC 61784-3-2 for the definition of this service.

4.1.8.9.2.4 SafetyClose Service

See IEC 61784-3-2 for the definition of this service.

4.1.9 Message and connection paths

4.1.9.1 Contents

The path shall specify the object element that is either the target of a connection request, or the destination of a message request. The path shall contain multiple segments which can indicate the route to the next node (in the case of multiple links), what to connect to or where to send a message in the target device. Each segment shall be comprised of a segment descriptor octet which specifies the segment type and segment information whose size is dependent on the segment type. The two types of path shall be

- padded paths;
- packed paths.

These two path formats (padded and packed) shall not be interchangeable. Usage of padded versus packed shall be dependent on the context of use. If no format is explicitly specified for a path, then padded format shall be used.

Each segment of a padded path shall be 16 bit word aligned. If a pad octet is required to achieve the alignment, then the segment shall specify the location of the pad octet. A packed path shall not contain the pad octets.

The possible segment types shall be as follows:

Port segment (where);
 Logical segment (what);
 Network segment (when or how);
 Symbolic segment;
 Data segment.

4.1.9.2 Segment type

Each segment of the path shall contain a segment type to indicate how the segment is to be interpreted. The segment type is contained in the first octet of the segment. The bits of the first octet of a path segment shall be numbered 0 to 7, where bit 0 shall be the least significant bit of the octet. Bits of the segment type are defined in Figure 9.

Segment Type Depends on segment Type

0	0	0	Port
0	0	1	Logical
0	1	0	Network
0	1	1	Symbolic
1	0	0	Data / Extended symbolic
1	0	1	Reserved
1	1	0	Reserved
1	1	1	Reserved

Figure 9 – Segment type

4.1.9.3 Port segment

The port segment shall indicate

- communication port through which to leave the node (expressed as a 16-bit number);
- link address of the next device in the path.

Figure 10 shows the overall structure of the port segment.

The bits 5-7 of the first octet shall be zero to indicate the port segment type. Bit 4, the node/slot address size bit, shall be set to 0 if the link address is one octet. Bit 4 shall be set to 1 if the link address is larger than one octet. If the link address is larger than one octet, its size in octets shall be in the second octet of the port segment.

Bits 0 – 3, the port identifier, shall indicate through which port to leave the node. The port identifier shall specify a port number or an escape to an extended port identifier when the module can support more than 14 ports. Port number 0 shall be reserved. Port number 1 shall only be used to represent a back-plane port. If the port identifier is 15, then a 16-bit field, called the extended port identifier, shall be the next part of the port segment; otherwise, the value of the port identifier shall be the port number.

The port number shall be followed by a link address whose format depends on the port type. If the link address is greater than one octet, then it shall be padded so that the entire port segment is an even number of octets. The octet specifying the size of the link address shall not include the pad octet. The pad octet shall be set to 0.

NOTE 1 For the common port types, the link address is a single octet. Other port types, such as TCP/IP encapsulation, may require a larger link address (see 4.3 and Clause 11).

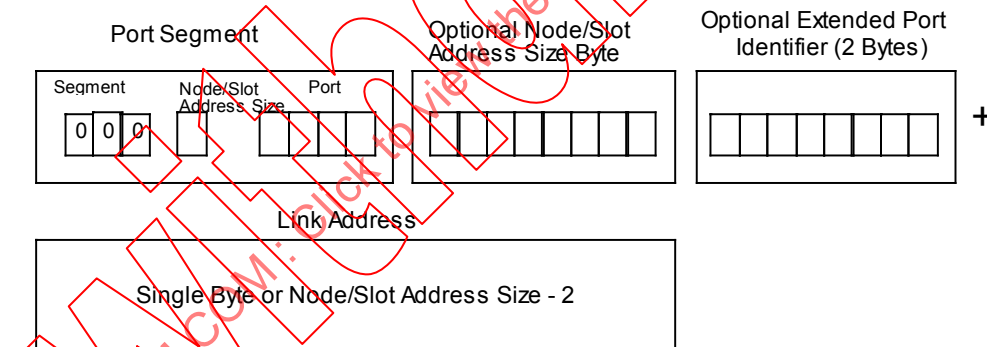


Figure 10 – Port segment

The Extended Port Identifier format of the Port Segment shall only be used when there are more than 14 ports possible on the device. The Port Segment shall always be packed into the smallest Port Segment format possible. Examples of possible port segments are as shown in Table 125.

Table 125 – Possible port segment examples

Port segment	Notes
[02][06]	Port 2, MAC ID 6
[15][0E][31][33][31][2E][31][35][31][2E][31][33][32][2E][35][35]	Multi-Octet Address for Ethernet Port 5, Link Address 131.151.132.55 (IP Address). The Address is defined as a character array.
[0F][12][00][01]	Port 18, MAC ID 1

The link address portion of a TCP/IP connection path segment shall be encoded within the port segment as a string of ASCII characters. The string shall be one of the following forms:

- IP address in dot notation, for example “130.151.132.55” (see IETF RFC 1117 for the format of IP addresses);
- IP address in dot notation, followed by a “:” separator, followed by the TCP port number to be used at the specified IP address;
- Host name, for example “plc.type2.org”. The host name shall be resolved via a DNS request to a name server (see IETF RFC 1035 for information on host names and name resolution);
- Host name, followed by a “:” separator, followed by the TCP port number to be used at the specified host.

The port number shall be represented in either hex or decimal. Hex shall be indicated by a leading “0x”. When a port number is specified, it shall be used rather than the standard port number used for the encapsulation protocol (0xAF12). Only port 0xAF12 is guaranteed to be available in a CP 2/2 compliant device.

NOTE 2 Other TCP port numbers may be implemented; however, this specification does not provide a mechanism to determine which TCP port numbers are supported by a device. The use of other TCP port numbers is therefore discouraged. The guaranteed TCP port number, 0xAF12, has been reserved with the Internet Assigned Numbers Authority (IANA) for use by the encapsulation protocol.

Since port segments shall be word-aligned, a pad octet may be required at the end of the string. The pad octet shall be 0x00, and shall not be counted in the Optional Address Size field of the port segment.

NOTE 3 Examples of port segments for TCP/IP are shown in Table 126 below.

Table 126 – TCP/IP link address examples

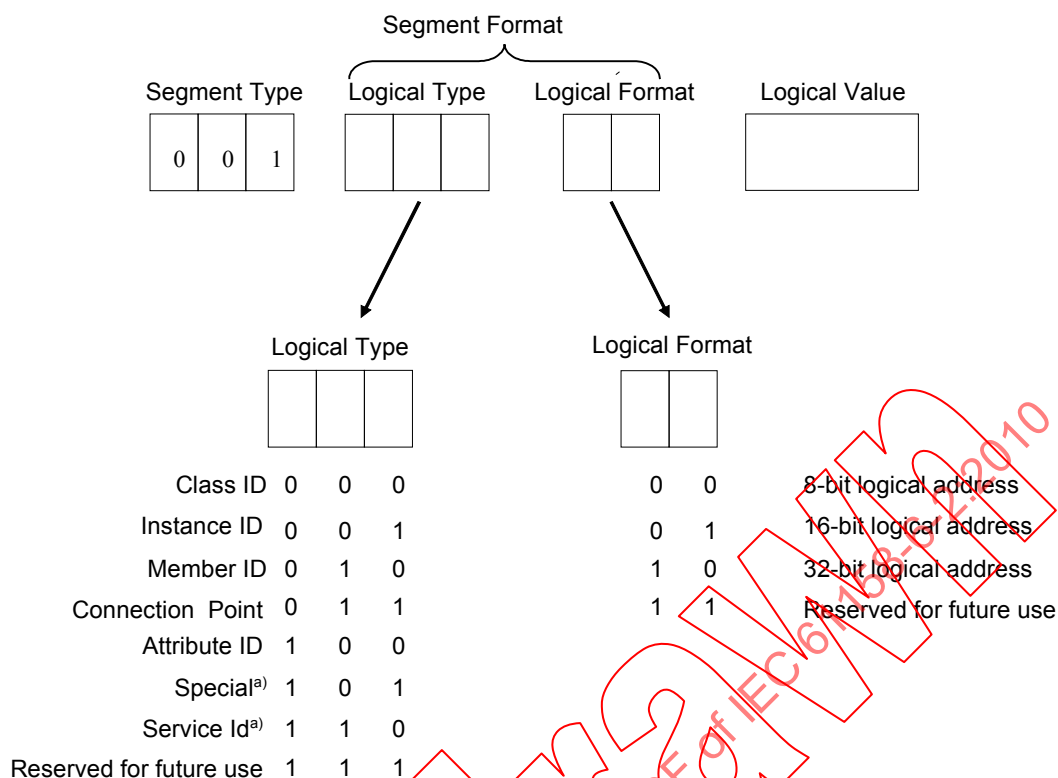
Port segment	IP address	Notes
[12][0D] [31][33][30][2E] [31][35][31][2E] [31][33][32][2E][31][00]	130.151.132.1	Multi-octet address for port 2, 13 octet string plus a pad octet
[13][12] [70][6C][63][2E] [63][6F][6E][74][72][6F][6C][6E][65][74][2E] [6F][72][67]	plc.type2.org	Multi-octet address for port 3, 18 octet string, no pad octet
[16][15] [31][33][30][2E] [31][35][31][2E] [31][33][32][2E] [35][35][3A] [30][78][33][32][31][30][00]	130.151.132.55:0x3210	Multi-octet address for port 6, 21 octet string plus a pad octet
[15][17] [70][6C][63][2E] [63][6F][6E][74][72][6F][6C][6E][65][74][2E] [6F][72][67][3A] [39][38][37][36][00]	plc.type2.org:9876	Multi-octet address port 5, 32 octet string plus a pad octet

4.1.9.4 Logical segment

4.1.9.4.1 Logical segment structure

The logical segment selects a particular addressable entity within a device (for example, object class, object instance, and object attribute). When the logical segment is included within a Packed Path, the Logical Value shall be appended to the segment type octet with no pad in between. When the logical segment is included within a Padded Path, the 16-bit and 32-bit logical formats shall have a pad inserted between the segment type octet and the Logical Value (the 8-bit format is identical to the Packed Path). The pad octet shall be set to zero.

Figure 11 specifies the encoding of a logical segment.



^{a)}The Special and Service ID Logical Types do not use the logical addressing definition for the Logical Format.

Figure 11 – Logical segment encoding

The 8-bit and 16-bit logical address formats are allowed for use with all Logical Types.

The 32-bit logical address format is only allowed for the logical Instance ID and Connection Point types. It is not allowed for any other Logical Type (reserved for future use).

The Class ID specifies a type of object within a device. The Instance ID is an integer assigned to an object when it is created. Instance 0, called the class instance, specifies the class itself. The attributes of an object are an enumerated list of externally visible values. The attribute ID specifies which attribute of an object is selected. Complex attributes are composed of members. The member ID specifies which member of the attribute.

The Connection Point Logical Type provides additional addressing capabilities beyond the standard Class ID/Instance ID/Attribute ID/Member ID object address. Object classes shall define when and how this addressing component is utilized (for example, a sub-instance of an Assembly object).

NOTE IEC 61158-5-2:2010 describes the object model including the definition of class, instance, attribute, connection point, and members of a library of common objects.

The Service ID Logical Type has the following definition for the Logical Format:

- 0 0 8-Bit Service ID Segment (0x38)
- 0 1 Reserved for future use (0x39)
- 1 0 Reserved for future use (0x3A)
- 1 1 Reserved for future use (0x3B)

The Special Logical Type has the following definition for the Logical Format:

- 0 0 Electronic Key Segment (0x34)
- 0 1 Reserved for future use (0x35)
- 1 0 Reserved for future use (0x36)
- 1 1 Reserved for future use (0x37)

4.1.9.4.2 Electronic Key Segment

The electronic key segment shall be used to verify the type of target or router device. The key may be included as the first logical segment in a connection path and shall be checked by the Message Router of the receiving node against the values contained in instance 1 of its Identity object before any additional address checks are made. On a multi-hop message, sent to a remote link via intermediate routers, multiple key segments may appear. The format of the electronic key segment shall be as shown in Table 127.

Table 127 – Electronic key segment format

Parameter	Format	Value
segment_type	USINT	always 0x34
electronic_key_type	USINT	always 0x04
vendor_ID	UINT	these correspond to the values in instance 1 of the identity object
product_type	UINT	
product_code	UINT	
major_revision : 7	USINT	
compatible_match : 1	USINT	
minor_revision	USINT	

The `segment_type` for a key segment shall be 0x34. The `electronic_key_type` shall determine the format of a specific key segment and shall be set to 0x04. All other values for the `electronic_key_type` shall be reserved.

The `vendor_ID` shall specify the device vendor, or zero if no specific vendor is required.

The `product_type` shall specify a class of products such as digital input or analogue outputs. The `product_type` shall be ignored when it is set to zero. The `product_code` shall be vendor specific with each vendor having a unique code. The `product_code` shall be ignored when set to zero.

The `major_revision` and `compatible_match` fields shall be contained in the least significant 7 bits and most significant bit of the same octet, respectively.

The `major_revision` shall specify the functionality level of a device. The `major_revision` shall be ignored when set to zero.

The `compatible_match` shall modify the key matching function. If the bit is clear (= 0), then any non-zero `vendor_ID`, `product_type`, `product_code`, `major_revision`, and `minor_revision` shall match exactly. If the bit is set (= 1), the device may accept the key for any device which it can emulate. The level of emulation shall be product specific.

The `minor_revision` shall specify the revision of a device that does not effect network-visible behaviour. A value of zero shall specify that the originator of the request accepts any minor revision.

NOTE 2 IEC 61158-5-2:2010 describes the Identity object and the rules for updating the revision fields.

4.1.9.4.3 Logical segments examples

Table 128 shows examples of logical segments.

Table 128 – Logical segments examples

First octet	Logical segment name	Padded format examples (as transmitted)	Packed format examples (as transmitted)	Notes
0x20	8-bit class	[20][04]	[20][04]	class 0x04 (Assembly object)
0x21	16-bit class	[21][00][34][12]	[21][34][12]	class 0x1234
0x24	8-bit instance	[24][04]	[24][04]	instance 0x04
0x25	16-bit instance	[25][00][34][12]	[25][34][12]	instance 0x1234
0x28	8-bit member	[28][04]	[28][04]	member 0x04
0x29	16-bit member	[29][00][34][12]	[29][34][12]	member 0x1234
0x2A	32-bit member			
0x2C	8-bit connection point	[2C][04]	[2C][04]	connection point 0x04
0x2D	16-bit connection point	[2D][00][34][12]	[2D][34][12]	connection point 0x1234
0x30	8-bit attribute	[30][04]	[30][04]	attribute 0x04
0x31	16-bit attribute	[31][00][34][12]	[31][34][12]	attribute 0x1234
0x34	electronic key	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	vendor ID = 0x0012 product type = 0xBEEF product code = 0x0B0E major revision = 0x03 minor revision = 0x01 accept compatible keys = yes

4.1.9.5 Network segment

4.1.9.5.1 Network segment structure

The segment type (first octet) of a network segment shall be in the range 0x40 through 0x5F as shown in Table 129 (the most significant bits shall be 010). The network segment shall be used to specify network parameters which may be required by a node to transmit a message across a network. The network segment shall immediately precede the port segment of the device to which it applies. In other words, the network segment shall be the first item in the path that the device receives.

Table 129 – Network segments

First octet	Network segment name
0x40	reserved
0x41	schedule segment
0x42	fixed tag segment
0x43	production inhibit time segment
0x44 – 0x4F	reserved
0x050	safety segment
0x051 – 0x05E	reserved
0x5F	extended network segment

4.1.9.5.2 Schedule Segment

The segment type of the schedule network segment shall be 0x41. The schedule network segment shall specify

- a multiplier that, when multiplied by the NUT, gives the CM_API (actual packet interval) of the scheduled transport;
- a phase that determines on which values of the `Moderator.interval_count` to transmit.

NOTE 1 The data-link layer defines the `Moderator.interval_count`, see IEC 61158-4-2:2010.

This segment shall be included in the path to a device so that each intermediary node can schedule link transmit time for subsequent scheduled traffic. The multiplier shall be one of 1, 2, 4, 8, 16, 32, 64 or 128. The phase shall be in the range 0 through (multiplier – 1). The second octet of the schedule network segment shall encode both the multiple and phase by adding them together.

NOTE 2 If a transport produces every 64th NUT starting on `interval_count` = 17, then the encoded value is 17 + 64 = 81.

An encoded value of zero shall specify that the transport has not yet been given permission to use the scheduled priority. If a node receives a request for a scheduled connection in which either no schedule network segment exists, or the schedule network segment exists but the encoded value is 0, the node shall return general status = 0x01, and extended status = 0x0317.

NOTE 3 A connection originator is given permission to use the scheduled priority through the Scheduling object (IEC 61158-5-2:2010).

4.1.9.5.3 Fixed Tag Segment

The segment type of the fixed tag network segment shall be 0x42. The fixed tag network segment shall specify the fixed tag which is to be used when sending an unconnected message. This segment subtype shall precede the port segment within the path. If the fixed tag segment is not present, then the default fixed tag shall be used.

NOTE 4 The fixed tag segment can be used within the path of the `Unconnected_Send` service (see 4.1.5.6) of the Connection Manager to send requests to the Keeper object via the Management UCMM (see IEC 61158-5-2:2010).

4.1.9.5.4 Production Inhibit Time Segment

The segment type of the production inhibit time segment shall be 0x43. The second octet shall specify the production inhibit time, which is the minimum time, in milliseconds, between successive transmissions of connected data for the specified connection.

NOTE 5 For example, if a production inhibit time of 10 milliseconds is specified, new data shall be sent no sooner than 10 milliseconds after the previous data.

The production inhibit time segment shall be used only for change-of-state transport class 1 connections over TCP/IP links, to place a limit on the packet rate. The production inhibit time segment may be sent to an end or intermediate device.

When the production inhibit time segment is used, the RPI shall determine when to re-send data. The RPI shall be larger than the production inhibit time. If the RPI is smaller than the production inhibit time, the `Forward_Open` response shall be returned with error (status 0x01, extended status 0x11B). If the production inhibit time segment is omitted, the producer shall use a default production inhibit time of 1/4 the RPI. If the production inhibit time is 0, there shall be no limit on how fast data may be sent on the connection.

NOTE 6 The method for transmitting APDUs over Ethernet-TCP/IP is specified in 4.3 and Clause 11.

4.1.9.5.5 Safety Segment

See IEC 61784-3-2 for format.

4.1.9.5.6 Extended Network Segment

The extended network segment allows for the definition of additional network segment subtypes. The first word of data is the extended network segment subtype. The structure of the extended network segment is shown in Figure 12.

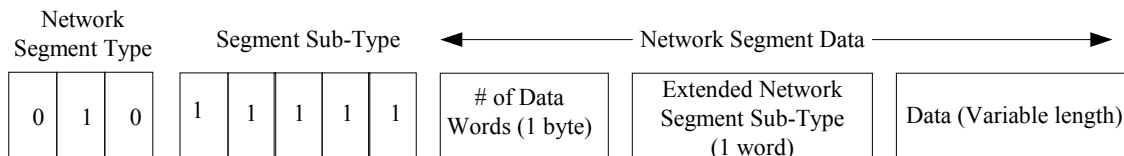


Figure 12 – Extended network segment

Each extended subtype defines the data that follows. The extended subtypes are enumerated in Table 130.

Table 130 – Extended subtype definitions

Extended Subtype Value	Extended Subtype Definition
0 – 65 535	Reserved for future use

4.1.9.6 Symbolic segment

The range of segment types reserved for symbolic segments shall be 0x60 through 0x7F.

4.1.9.7 Data segment

4.1.9.7.1 Data segment structure

The segment type (first octet) of a data segment shall be in the range 0x80 through 0x8F (the most significant bits shall be 1000). This standard only defines the format of the simple data segment with a segment type of 0x80; all other data segment types shall be reserved (0x81 through 0x8F).

4.1.9.7.2 Simple Data Segment

The segment type of the simple data segment shall be 0x80. The second octet shall represent the number of 16-bit words of variable length data. Following the second octet shall be the variable length data. A path may contain more than one simple data segment as shown in Table 131.

Table 131 – Data segment

Parameter	Format	Value
segment_type	USINT	always 0x80
segment_size	USINT	size of the data[] array in 16-bit words
data[]	UINT	

The data can be configuration information for an object, additional parameters necessary for the connection, or any other information. The data segment shall not be interpreted by any other device in the path, so only the originating and target applications need agree on its contents.

4.1.9.8 Extended symbol segment

4.1.9.8.1 Extended symbol segment structure

The segment type (first octet) of an extended symbol segment shall be in the range 0x90 through 0x9F (the most significant bits shall be 1001). This STANDARD only defines the format of the extended symbol segments 0x91. All other extended symbol segment types shall be reserved (0x90 and 0x92 through 0x9F).

4.1.9.8.2 ANSI Extended Symbol Segment

The segment type of the ANSI extended symbol segment shall be 0x91. The second octet shall represent the number of characters (8-bit) in the symbol. Following the second octet shall be the variable length symbol as shown in Table 132.

Table 132 – ANSI_Extended_Symbol segment

Parameter	Format	Value
segment_type	USINT	always 0x91
symbol_size	USINT	size of the symbol[] array
symbol[]	USINT	
pad	USINT	only present if symbol_size is odd, always set to zero

The `symbol_size` shall be the size of the `symbol[]` array in octets and shall not be zero. The `symbol_size` shall not count the `pad` octet if it is included.

4.1.10 Class, attribute and service codes

4.1.10.1 Code ranges

4.1.10.1.1 Defined ranges

There shall be three categories for address ranges of Class IDs, Attribute IDs and Service Codes as specified in Table 133.

Table 133 – Addressing categories

This category	Refers to
Open	A value which has the same meaning for all implementers. All object classes and services defined in IEC 61158-5-2:2010 fall under this category.
Vendor-specific	A range of values specific to the vendor of a device. These are used by vendors to extend their devices beyond the available <i>Open</i> options. A vendor internally manages the use of values within this range. Applies to object classes, attributes, and services.
Object-class specific	A range of values whose meaning is defined by an object class. This range applies to Service Code definitions.
NOTE Open values are assigned by ODVA, Inc.	

The Class ID, Attribute ID and Service code values shall be as defined in Table 134 through Table 136.

4.1.10.1.2 Class code ID ranges

The Class ID values shall be as shown in Table 134. Class Code = 0x00 is reserved and shall not be used.

Table 134 – Class code ID ranges

Range (hex)	Range (decimal)	Meaning	Quantity
00	00	Reserved	1
01 - 63	01 - 99	Open	99
64 - C7	100 - 199	Vendor Specific	100
C8 - EF	200 - 239	Reserved	40
F0 - 2FF	240 - 767	Open	528
300 - 4FF	768 - 1 279	Vendor Specific	512
500 - FFFF	1 280 - 65 535	Reserved	64 256

4.1.10.1.3 Attribute ID ranges

The Attribute ID values shall be as shown in Table 135. Attribute ID = 0x00 is reserved and shall not be used.

Table 135 – Attribute ID ranges

Range (hex)	Range (decimal)	Meaning	Quantity
00 – 63	00 – 99	Open	100
64 – C7	100 – 199	Vendor Specific	100
C8 – FF	200 – 255	Reserved	56
100 – 2FF	240 – 767	Open	512
300 - 4FF	768 – 1 279	Vendor Specific	512
500 – 8FF	1 280 – 2 303	Open	1 024
900 – CFF	2 304 – 3 327	Vendor Specific	1 024
D00 - FFFF	3 328 – 65 535	Reserved for future use	62 208

4.1.10.1.4 Service code ranges

The **Service code** shall be a unique hexadecimal value assigned to each service. The Service Code values shall be as shown in Table 136. Service Code = 0x00 is reserved and shall not be used. The object-specific service codes shall be unique only within the class which define them.

Table 136 – Service code ranges

Range (hex)	Range (decimal)	Meaning	Quantity
00	00	Reserved	1
01 - 31	01 - 49	Open. These are referred to as <i>Common Services</i> . They are defined in Type 2 clauses in IEC 61158-5-2:2010	49
32 - 4A	50 - 74	Vendor Specific	25
4B - 63	75 - 99	Object Class Specific	25
64 - 7F	100 - 127	Reserved for future use	28
80 - FF	128 - 255	Reserved for response messages	128

4.1.10.2 Code definitions

4.1.10.2.1 Communication object classes

Table 137 defines the class codes for the object classes. All other class codes listed in within the “open” range and not listed in Table 137 shall be reserved. The requirements for these objects are defined in IEC 61158-5-2:2010.

Table 137 – Class codes

Class code (hex)	Class code (decimal)	Object name
0x01	1	Identity object
0x02	2	Message Router object
0x03	3	DeviceNet object ^a
0x04	4	Assembly object
0x05	5	Connection object
0x06	6	Connection Manager object
0x0F	15	Parameter object
0x2B	43	Acknowledge Handler object
0x43	67	Time Sync object
0xF0	240	ControlNet object ^a
0xF1	241	Keeper object ^a
0xF2	242	Scheduling object ^a
0xF3	243	Connection Configuration object ^a
0xF5	245	TCP/IP Interface object ^a
0xF6	246	Ethernet Link object ^a

^a This object class is part of system management.

4.1.10.2.2 Predefined class attributes

Seven predefined Class Attribute IDs shall be reserved, and these predefined/reserved class attributes shall have the definitions listed in Table 138. Because these attributes are reserved, class Attribute ID numbers 1 through 7 shall always be reserved. Therefore, if a class attribute is added to an object class specification, it shall start with Attribute ID #8.

Table 138 – Reserved class attributes for all object class definitions

Attribute ID	Name	Data type	Semantics of values
1	Revision	UINT	The starting value assigned to this attribute is one (1). If updates that require an increase in this value are made, then the value of this attribute increases by one (1).
2	Max Instance	UINT or UDINT	The class definition shall define the data type used.
3	Number of Instances	UINT or UDINT	The class definition shall define the data type used.
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	

Attribute ID	Name	Data type	Semantics of values
	Optional services	ARRAY of UINT	
6	Maximum ID Number Class Attributes	UINT	
7	Maximum ID Number Instance Attributes	UINT	

4.1.10.2.3 OM_Service codes

Codes of the common services for the deterministic control network objects shall be as shown in Table 139.

NOTE Table 139 lists the codes and names of the common services while IEC 61158-5-2:2010 provides a general description of each service.

Table 139 – Common services list

Common service code	Common service name
0x00	reserved
0x01	Get_Attribute_All
0x02	Set_Attribute_All
0x03	Get_Attribute_List
0x04	Set_Attribute_List
0x05	Reset
0x06	Start
0x07	Stop
0x08	Create
0x09	Delete
0x0A – 0x0C	reserved
0x0D	Apply_Attributes
0x0E	Get_Attribute_Single
0x0F	reserved
0x10	Set_Attribute_Single
0x11	Find_Next_Object_Instance
0x12 – 0x14	reserved
0x15	Restore
0x16	Save
0x17	NOP (No operation)
0x18 – 0x1B	reserved
0x1C	Group_Sync
0x1D – 0x31	reserved

Codes of the object specific services for the Acknowledge Handler object shall be as shown in Table 140.

Table 140 – Acknowledge Handler object specific services list

Object specific service code	Service name
0x4B	Add_AckData_Path
0x4C	Remove_AckData_Path

Code of the object specific service for the Parameter object shall be as shown in Table 141.

Table 141 – Parameter object specific services list

Object specific service code	Service name
0x4B	Get_Enum_String

4.1.10.2.4 CM_Service codes

Codes of the services specific to the Connection Manager shall be as shown in Table 142.

NOTE Table 142 lists the codes and names of the services while IEC 61158-5-2:2010 provides a general description of each service.

Table 142 – Services specific to Connection Manager

Service code	Service name
0x4E	Forward_Close
0x52	Unconnected_Send
0x54	Forward_Open
0x56	Get_Connection_Data
0x57	Search_Connection_Data
0x5A	Get_Object_Owner
0x5B	Large_Forward_Open

4.1.10.2.5 CO_Service codes

Codes of the services specific to the Connection object shall be as shown in Table 143.

NOTE Table 143 lists the codes and names of the services while IEC 61158-5-2:2010 provides a general description of each service.

Table 143 – Services specific to Connection object

Service code	Service name
0x4B	Connection_Bind
0x4C	Producing_Application_Lookup
0x4E	SafetyClose
0x54	SafetyOpen

4.1.10.3 Device types

In order to allow interoperability and interchangeability, a device type shall be used to identify similar devices which

- exhibit the same behaviour;

- produce and/or consume the same set of data;
- contain the same set of configurable attributes.

The formal definition of this information is known as a Device Profile : all devices with the same device type number shall meet the requirements specified in the Device Profile for that device type.

“Device Type” is a required instance attribute of the Identity object as described in IEC 61158-5-2:2010.

Device types shall be either publicly defined or vendor specific. Table 144 reveals the numbering scheme to be used for device type numbering.

Table 144 – Device type numbering

Range (hex)	Type	Quantity
00 - 63	Publicly Defined – Reserved	100
64 - C7	Vendor Specific	100
C8 - FF	Reserved	56
100 - 2FF	Publicly Defined – Reserved	512
300 - 4FF	Vendor Specific	512
500 - FFFF	Reserved	64 256

All publicly defined device types shall have the same meaning for all implementers and are reserved. The Generic Device type (type number = 0x00) shall define a device that does not fit into any of the defined device types.

NOTE The actual definition of publicly defined device types and corresponding Device Profiles is outside the scope of this standard. They are defined by ODVA, Inc.

Vendor specific device types may be developed and vendors need not publish them. Each vendor shall maintain their own vendor-specific device types and corresponding number allocation.

4.1.11 Error codes

4.1.11.1 CM errors

The error codes are returned with the response to a Connection Manager Service Request which resulted in an error. These error codes shall be used to help diagnose the problem with a Service Request. The error code shall be split into an 8 bit general status and one or more 16 bit words of extended status. Unless specified otherwise, only the first word of extended status shall be required. Additional words of extended status may be used to specify additional module specific debug information. All devices which originate messages shall be able to handle multiple words of extended status.

Table 145 provides a summary of the available error codes.

Table 145 – Connection Manager service request error codes

General Status	Extended Status	Explanation
0x00		Service completed successfully.
0x01	0x0000 – 0x00FF	Obsolete
0x01	0x0100	CONNECTION IN USE OR DUPLICATE FORWARD OPEN This extended status code shall be returned when an originator is trying to make a connection to a target with which the originator may have already established a connection (duplicate Forward_Open – see 4.1.5.3).
0x01	0x0101 – 0x0102	Reserved
0x01	0x0103	TRANSPORT CLASS AND TRIGGER COMBINATION NOT SUPPORTED A transport class and trigger combination has been specified which is not supported by the target. Routers shall not fail the connection based on the transport class and trigger combination. Only targets shall return this extended status code
0x01	0x0104 – 0x0105	Reserved
0x01	0x0106	OWNERSHIP CONFLICT The connection cannot be established since another connection has exclusively allocated some of the resources required for this connection. An example of this would be that only one exclusive owner connection can control an output point on an I/O Module. If a second exclusive owner connection (or redundant owner connection) is attempted, this error shall be returned. This extended status code shall only be returned by a target node.
0x01	0x0107	TARGET CONNECTION NOT FOUND This extended status code shall be returned in response to the forward_close request, when the connection that is to be closed is not found at the target node. This extended status code shall only be returned by a target node. Routers shall not generate this extended status code. If the specified connection is not found at the intermediate node, the close request shall still be forwarded using the path specified in the Forward_Close request.
0x01	0x0108	INVALID NETWORK CONNECTION PARAMETER This extended status code shall be returned as the result of specifying a connection type, connection priority, redundant owner or fixed/variable that is not supported by the target application. Only a target node shall return this extended status code.
0x01	0x0109	INVALID CONNECTION SIZE This extended status code is returned when the target or router does not support the specified connection size. This could occur at a target because the size does not match the required size for a fixed size connection. It could occur at a router if the requested size is too large for the specified network. An additional status may follow indicating the maximum connection size supported by the responding node. The additional status word is required when issued in response to the Large_Forward_Open.
0x01	0x010A – 0x010F	Reserved
0x01	0x0110	TARGET FOR CONNECTION NOT CONFIGURED This extended status code shall be returned when a connection is requested to a target application that has not been configured and the connection request does not contain a data segment for configuration (see 4.1.9.7). Only a target node shall return this extended status code.

General Status	Extended Status	Explanation
0x01	0x0111	RPI NOT SUPPORTED. This extended status code shall be returned if the device can not support the requested $O \Rightarrow T$ or $T \Rightarrow O$ RPI. This extended status code shall also be used if the connection time-out multiplier produces a time-out value that is not supported by the device or the production inhibit time is not valid.
0x01	0x0112	Reserved
0x01	0x0113	OUT OF CONNECTIONS Connection Manager cannot support any more connections. The maximum number of connections supported by the Connection Manager has already been created.
0x01	0x0114	VENDOR ID OR PRODUCT CODE MISMATCH The Product Code or Vendor Id specified in the electronic key logical segment does not match the Product Code or Vendor Id of the target device.
0x01	0x0115	PRODUCT TYPE MISMATCH The Product Type specified in the electronic key logical segment does not match the Product Type of the target device.
0x01	0x0116	REVISION MISMATCH The major and minor revision specified in the electronic key logical segment does not correspond to a valid revision of the target device.
0x01	0x0117	INVALID PRODUCED OR CONSUMED APPLICATION PATH The produced or consumed application path specified in the connection path does not correspond to a valid produced or consumed application path within the target application. This error could also be returned if a produced or consumed application path was required, but not provided by a connection request.
0x01	0x0118	INVALID OR INCONSISTENT CONFIGURATION APPLICATION PATH An application path specified for the configuration data does not correspond to a configuration application or is inconsistent with the consumed or produced application paths. For example the connection path specifies float configuration data while the produced or consumed paths specify integer data.
0x01	0x0119	NON-LISTEN ONLY CONNECTION NOT OPENED Connection request fails since there are no non-listen only connection types currently open. See IEC 61158-5-2:2010, 6.3.1.4.5, for a description of application connection types. The extended status code shall be returned when an attempt is made to establish a listen only connection type to a target, which has no non-listen only connection already established.
0x01	0x011A	TARGET OBJECT OUT OF CONNECTIONS The maximum number of connections supported by this instance of the target object has been exceeded. For example, the Connection Manager could support 20 connections while the target object can only support 10 connections. On the 11th Connection Request to the target object, this extended status code would be used to signify that the maximum number of connections already exist to the target object.
0x01	0x011B	RPI IS SMALLER THAN THE PRODUCTION INHIBIT TIME The Target to Originator RPI is smaller than the Target to Originator Production Inhibit Time.
0x01	0x011C – 0x0202	Reserved
0x01	0x0203	CONNECTION TIMED OUT This extended status code shall occur when a client tries to send a connected message over a connection that has been timed-out. This extended status code shall only occur locally at the producing node.

General Status	Extended Status	Explanation
0x01	0x0204	UNCONNECTED REQUEST TIMED OUT The Unconnected Request Timed Out error shall occur when the UCMM times out before a response is received. This may occur for an Unconnected_Send, Forward_Open, or Forward_Close service. This typically means that the UCMM has tried a link specific number of times using a link specific retry timer and has not received an acknowledgement or response. This may be the result of congestion at the destination node or may be the result of a node not being powered up or present. This extended status code shall be returned by the originating node or any intermediate node.
0x01	0x0205	PARAMETER ERROR IN UNCONNECTED REQUEST SERVICE For example, this shall be caused by a Connection Tick Time (see IEC 61158-5-2:2010, 6.2.3.2.1.7) and Connection time-out combination in an Unconnected_Send, Forward_Open, or Forward_Close service that is not supported by an intermediate node.
0x01	0x0206	MESSAGE TOO LARGE FOR UNCONNECTED_SEND SERVICE This shall be caused when the Unconnected_Send is too large to be sent out on a network.
0x01	0x0207	UNCONNECTED ACKNOWLEDGE WITHOUT RESPONSE The message was sent via the unconnected message service and an acknowledge was received but a data response message was not received.
0x01	0x0208 – 0x0300	Reserved
0x01	0x0301	NO BUFFER MEMORY AVAILABLE The extended status code shall occur when insufficient connection buffer memory is available in the target or any router devices. Routers and target nodes shall return this error.
0x01	0x0302	LINK TRANSMIT TIME NOT AVAILABLE FOR DATA This extended status code shall be returned by any device in the path that is a producer and can not allocate sufficient link transmit time for the connection on its link. This can occur at any node. This can only occur for connections that are specified as scheduled priority.
0x01	0x0303	NO CONSUMED CONNECTION ID FILTER AVAILABLE Any device in the path that contains a link consumer for the connection and does not have an available consumed_connection_id filter available shall return this extended status code.
0x01	0x0304	NOT CONFIGURED TO SEND SCHEDULED PRIORITY DATA If requested to make a connection that specifies scheduled priority, any device that is unable to send packets during the scheduled portion of the network update time interval shall return this extended status code. For example, on CP 2/1 this code shall be returned by a node whose MAC ID is greater than maximum scheduled node (SMAX).
0x01	0x0305	SCHEDULE SIGNATURE MISMATCH This extended status code shall be returned when the connection scheduling information in the originator device is not consistent with the connection scheduling information on the target network.
0x01	0x0306	SCHEDULE SIGNATURE VALIDATION NOT POSSIBLE This extended status code shall be returned when the connection scheduling information in the originator device can not be validated on the target network. For example, on CP 2/1 this code shall be returned when there is no Keeper in the master state.
0x01	0x0307 – 0x0310	Reserved
0x01	0x0311	PORT NOT AVAILABLE A Port specified in a Port Segment is Not Available or does not exist.

General Status	Extended Status	Explanation
0x01	0x0312	LINK ADDRESS NOT VALID Link Address specified in Port Segment Not Valid This extended status code is the result of a port segment that specifies a link address that is not valid for the target network type. This extended status code shall not be used for link addresses that are valid for the target network type but do not respond.
0x01	0x0313 – 0x0314	Reserved
0x01	0x0315	INVALID SEGMENT IN CONNECTION PATH Invalid Segment Type or Segment Value in Connection Path This extended status code is the result of a device being unable to decode the connection path. This could be caused by an unrecognized path type, a segment type occurring unexpectedly, or a myriad of other problems in the connection path.
0x01	0x0316	ERROR IN FORWARD CLOSE SERVICE CONNECTION PATH MISMATCH The connection path in the Forward_Close service does not match the connection path in the connection being closed.
0x01	0x0317	SCHEDULING NOT SPECIFIED Either the Schedule Network Segment was not present or the Encoded Value in the Schedule Network Segment is invalid (0).
0x01	0x0318	LINK ADDRESS TO SELF INVALID Under some conditions (depends on the device), a link address in the Port Segment which points to the same device (loopback to yourself) is invalid.
0x01	0x0319	SECONDARY RESOURCES UNAVAILABLE In a dual-chassis redundant system, a connection request that is made to the primary system shall be duplicated on the secondary system. If the secondary system is unable to duplicate the connection request, then this extended status code shall be returned.
0x01	0x031A	RACK CONNECTION ALREADY ESTABLISHED A request for a module connection has been refused because part of the corresponding data is already included in a rack connection.
0x01	0x031B	MODULE CONNECTION ALREADY ESTABLISHED A request for a rack connection has been refused because part of the corresponding data is already included in a module connection.
0x01	0x031C	MISCELLANEOUS This extended status is returned when no other extended status code applies for a connection related error.
0x01	0x031D	REDUNDANT CONNECTION MISMATCH This extended status code shall be returned when the following fields do not match when attempting to establish a redundant owner connection to the same target path: O->T_RPI; O->T_connection_parameters; T->O_RPI; T->O_connection_parameters; xport_type_and_trigger.
0x01	0x031E	NO MORE USER CONFIGURABLE LINK CONSUMER RESOURCES AVAILABLE IN THE PRODUCING MODULE A target shall return this extended status when the configured number of consumers for a producing application are already in use.

General Status	Extended Status	Explanation
0x01	0x031F	NO MORE USER CONFIGURABLE LINK CONSUMER RESOURCES AVAILABLE IN THE PRODUCING MODULE A target shall return this extended status when there are no consumers configured for a producing application to use.
0x01	0x0320 – 0x07FF	Vendor specific
0x01	0x0800	Network link in path to module is offline
0x01	0x0801 – 0x080F	Reserved
0x01	0x0810	NO TARGET APPLICATION DATA AVAILABLE This extended status code is returned when the target application does not have valid data to produce for the requested connection. Only the target side of a connection shall return this extended status code.
0x01	0x0811	NO ORIGINATOR APPLICATION DATA AVAILABLE This extended status code is returned when the originator application does not have valid data to produce for the requested connection. Only the originator side of a connection shall indicate this extended status code.
0x01	0x0812	NODE ADDRESS HAS CHANGED SINCE THE NETWORK WAS SCHEDULED A router on a scheduled network (e.g. GP 2/1) has a different node address than the value configured in the connection originator.
0x01	0x0813	NOT CONFIGURED FOR OFF-SUBNET MULTICAST A multicast connection has been requested between a producer and a consumer that are on different subnets, and the producer is not configured for off-subnet multicast.
0x01	0x0814	INVALID PRODUCE/CONSUME DATA FORMAT Information in the data segment indicates that the format of the produced and/or consumed data is not valid.
0x01	0x0815 – 0xFCFF	Reserved
0x09	Index to Element	ERROR IN DATA SEGMENT. This general status code shall be returned when there is an error in the data segment of a forward open. The Extended Status shall be the index to where the error was encountered in the Data Segment (see 4.1.9.7).
0x0C	Optional	OBJECT STATE ERROR This general status code shall be returned when the state of the target object of the connection prevents the service request from being handled. The Extended Status reports the object's present state. The extended status is optional. For example, a target (application) object of the connection may need to be in an edit mode before attributes can be set. This is different from a service being rejected due to the state of the device.
0x10	Optional	DEVICE STATE ERROR This general status code shall be returned when the state of the device prevents the service request from being handled. The Extended Status reports the device's present state. The extended status is optional. For example, a controller may have a key switch which when set to the "hard run" state causes Service Requests to several different objects to fail (i.e. program edits). This general status code would then be returned.
0x13	None	Not enough data. The service did not supply enough data to perform the specified operation.
0x15	None	Too much data. The service supplied more data than was expected.

General Status	Extended Status	Explanation
NOTE 1 The word "n/a" in the Extended Status Column is used to signify that there is no additional Extended Status which is required to be returned for the particular General Status Code.		
NOTE 2 The word "optional" in the Extended Status Column is used to signify that if Extended Status information is used, then the first word of that extended status is already defined and user defined extended status shall begin with the second word of extended status.		

4.1.11.2 OM errors

4.1.11.2.1 General status format

General status used in the service primitives is specified in Table 146.

Table 146 – General status codes

Status code	Name	Description and meaning of status code
0x00	Success	Service was successfully performed by the object specified
0x01	Connection failure	A connection related service failed along the connection path
0x02	Resource unavailable	Resources needed for the object to perform the requested service were unavailable
0x03	Invalid parameter value	See status code 0x20, which is the preferred value to use for this condition
0x04	Path segment error	The path segment identifier or the segment syntax was not understood by the processing node. Path processing shall stop when a path segment error is encountered
0x05	Path destination unknown	The path is referencing an object class, instance or structure element that is not known or is not contained in the processing node. Path processing shall stop when a path destination unknown error is encountered
0x06	Partial transfer	Only part of the expected data was transferred
0x07	Connection lost	The messaging connection was lost
0x08	Service not supported	The requested service was not implemented or was not defined for this class or object instance
0x09	Invalid attribute value	Invalid attribute data detected
0x0A	Attribute list error	An attribute in the Get_Attribute_List or Set_Attribute_List response has a non zero status
0x0B	Already in requested mode/state	The object is already in the mode/state being requested by the service
0x0C	Object state conflict	The object cannot perform the requested service in its current mode/state
0x0D	Object already exists	The requested instance of object to be created already exists
0x0E	Attribute not settable	A request to modify a non-modifiable attribute was received
0x0F	Privilege violation	A permission/privilege check failed
0x10	Device state conflict	The device's current mode/state prohibits the execution of the requested service
0x11	Response data too large	The data to be transmitted in the response buffer is larger than the allocated response buffer
0x12	Fragmentation of a primitive value	The service specified an operation that is going to fragment a primitive data value, i.e. half a REAL data type
0x13	Not enough data	The service did not supply enough data to perform the specified operation
0x14	Attribute not supported	The attribute specified in the request is not supported
0x15	Too much data	The service supplied more data than was expected

Status code	Name	Description and meaning of status code
0x16	Object does not exist	The object specified does not exist in the device
0x17	Service fragmentation sequence not in progress	The fragmentation sequence for this service is not currently active for this data
0x18	No stored attribute data	The attribute data of this object was not saved prior to the requested service
0x19	Store operation failure	The attribute data of this object was not saved due to a failure during the attempt.
0x1A	Routing failure, request packet too large for network	The service request packet was too large for transmission on a network in the path to the destination. The routing device was forced to abort the service
0x1B	Routing failure, response packet too large for network	The service response packet was too large for transmission on a network in the path from the destination. The routing device was forced to abort the service
0x1C	Missing attribute list entry data	The service did not supply an attribute in a list of attributes that was needed by the service to perform the requested behaviour
0x1D	Invalid attribute value list	The service is returning the list of attributes supplied with status information for those attributes that were invalid
0x1E	Embedded service error	An embedded service resulted in an error
0x1F	Vendor specific error	A vendor specific error has been encountered. The extended status code field of the error response defines the particular error encountered. Use of this general status code should only be performed when none of the status codes presented in this table or within an object class definition accurately reflect the error
0x20	Invalid parameter	A parameter associated with the request was invalid. This code is used when a parameter does not meet the requirements of this specification and/or the requirements defined in an application object specification
0x21	Write once value or medium already written	An attempt was made to write to a write once medium (e.g. WORM drive, PROM) that has already been written, or to modify a value that cannot be changed once established
0x22	Invalid Response Received	An invalid response is received (e.g. response service code does not match the request service code, or response message is shorter than the minimum expected reply size). This status code can serve for other causes of invalid responses
0x23	Buffer overflow	The message received is larger than the receiving buffer can handle. The entire message was discarded.
0x24	Message format error	The format of the received message is not supported by the server.
0x25	Key Failure in path	The Key Segment which was included as the first segment in the path does not match the destination module. The object specific status shall indicate which part of the key check failed.
0x26	Path Size Invalid	The size of the path which was sent with the Service Request is either not large enough to allow the Request to be routed to an object or too much routing data was included.
0x27	Unexpected attribute in list	An attempt was made to set an attribute that cannot be set at this time.
0x28	Invalid Member ID	The Member ID specified in the request does not exist in the specified Class/Instance/Attribute.
0x29	Member not settable	A request to modify a non-modifiable member was received.
0x2A	Group 2 only server general failure	This status code may only be reported by CP 2/3 Group 2 Only servers with 4 octets or less code space and only in place of Service not supported, Attribute not supported and Attribute not settable
0x2B – 0xCF	Reserved	Reserved for future extensions

Status code	Name	Description and meaning of status code
0xD0 – 0xFF	Reserved for object class and service errors	This range of status codes shall be used to indicate object class specific errors. Use of this range should only be performed when none of the status codes presented in this table accurately reflect the error that was encountered

NOTE IEC 61158-5-2:2010 contains more detail on the service response general status codes for each common service.

4.1.11.2.2 Extended status format

The `MR_response_Header` contains a parameter called `Extended_status[]` which is labeled extended status data.

Actual usage and definition of this extended status is dependant on the object class or service: each object class defines its own extended status values and value ranges (including the vendor specific ones).

4.1.11.2.3 Object-specific general and extended status format

4.1.11.2.3.1 General

The following clauses specify the format of object-specific general and extended status.

4.1.11.2.3.2 Identity object status format

Table 147 specifies the format of object-specific general and extended status for the Identity object.

Table 147 – Identity object status codes

General status code	8-bit associated extended status codes	Status Name	Description of Status
0x00 – 0xCF		General status codes	Defined in 4.1.11.2.1
	0x00 – 0xEE		Reserved extended status codes
	0xF0 – 0xFE	Vendor specific	Vendor specific extended status codes
	0xFF		Used with all general status codes when required and no other extended status code is assigned
0xD0		Hardware diagnostic	Device self testing and hardware diagnostic conditions
	0x00		Reserved
	0x01		Checksum (or CRC) error – Code space/ROM – Boot section
	0x02		Checksum (or CRC) error – Code space/ROM – Application section
	0x03		Checksum (or CRC) error – NV (flash/EEPROM) memory
	0x04		Invalid non-volatile (NV) memory – Configuration bad
	0x05		Invalid non-volatile (NV) memory – No configuration established
	0x06		RAM memory bad – The RAM memory in the device was determined to be experiencing inoperative cells
	0x07		ROM/Flash memory bad
	0x08		Flash/EEPROM (NV) Memory Bad
	0x09		Interconnect wiring error / signal path problem
	0x0A		Power problem – Over current

General status code	8-bit associated extended status codes	Status Name	Description of Status
	0x0B		Power problem – Over voltage
	0x0C		Power problem – Under voltage
	0x0D		Internal sensor problem
	0x0E		System clock fault
	0x0F		Hardware configuration does not match NV configuration
	0x10		Watchdog disabled/idle
	0x11		Watchdog timer expired
	0x12		Device over temperature
	0x13		Ambient temperature outside of operating limits
	0x14 – 0xEF	(reserved)	Reserved
	0xF0 – 0xFE		Vendor specific extended status codes
	0xFF		Used with all general status codes when required and no other extended status code is assigned
0xD1		Device status/states	Device status events and conditions
	0x01		Power applied
	0x02		Device reset
	0x03		Device power loss
	0x04		Activated
	0x05		Deactivated
	0x06		Enter self-test state
	0x07		Enter standby state
	0x08		Enter operational state
	0x09		Non-specific minor recoverable fault detected
	0x0A		Non-specific minor unrecoverable fault detected
	0x0B		Non-specific major recoverable fault detected
	0x0C		Non-specific major unrecoverable fault detected
	0x0D		Fault(s) corrected
	0x0E		Ccv changed
	0x0F		Heartbeat interval changed
	0x10 – 0xEF	(reserved)	
	0xF0 – 0xFE	Vendor specific	Vendor specific
	0xFF		Used with all general status codes when required and no other extended status code is assigned
0xD2 – 0xEF		Object specific general status codes	Reserved – Not yet assigned
	0x00 – 0xFF	Reserved	
0xF0 – 0xFF		Vendor specific general status codes	A vendor specific error has been encountered. The extended status code field of the error response defines the particular error encountered. Use of this general status code should only be performed when none of the status codes presented in this table or within an object class definition accurately reflect the error
	0x00 – 0xFF	Vendor specific extended status codes	All extended status codes are available for association with each vendor specific general status code

4.2 Data abstract syntax specification

4.2.1 Transport format specification

The lower layers of open system architectures are concerned with the transport of user data among distributed functional units. In these layers, the user data may be regarded simply as a sequence of octets. However, application layer entities may manipulate the values of quite complex data types. To achieve independence between the application layer and lower layers, data types may be specified in an abstract syntax notation.

Supplementing the abstract syntax with one or more algorithms (called encoding rules) may determine the values of the lower layer octets which carry the application layer values. The combination of the abstract syntax with a single set of transfer rules produces a specific transfer syntax.

4.2.2 Abstract syntax notation

The data type definitions provided in this STANDARD shall be written in Abstract Syntax Notation One (ASN.1), as defined in the ISO/IEC 8824 series. These type definitions shall be a part of the ASN.1 module "Network DataTypes." The beginning ASN.1 statement indicating that these definitions are in this module is:

```
control network DataTypes DEFINITIONS ::= BEGIN
```

and the closing ASN.1 statement shall be the keyword "END".

The abstract definitions that follow shall comprise the set of control network data types. In addition, provision is made to extend or derive new data types based on existing defined types, and to include those in a "type dictionary."

4.2.3 Control network data specification

The notation [typeId] for directly derived, enumerated, subrange and structured bit string data shall mean that the tag shall be taken from the "type" field in the corresponding VariableDictionaryEntry.

```
Network Data ::= CHOICE {ElementaryData, DerivedData}
ElementaryData ::= CHOICE {
    BOOL,
    FixedLengthInteger,
    FixedLengthReal,
    AnyTime,
    AnyDate,
    AnyString,
    FixedLengthBitString,
    EPATH}
DerivedData ::= CHOICE {
    DirectlyDerivedData,
    EnumeratedData,
    SubrangeData,
    StructuredBitStringData,
    ARRAY,
    STRUCT,
    FunctionBlockData}
DirectlyDerivedData ::= [typeId] NetworkData
EnumeratedData ::= [typeId] USINT
SubrangeData ::= [typeId] FixedLengthInteger
StructuredBitStringData ::= [typeId] FixedLengthBitString
FixedLengthInteger ::= CHOICE {SignedInteger, UnsignedInteger}
```

```

SignedInteger ::= CHOICE {SINT, INT, DINT, LINT}
UnsignedInteger ::= CHOICE {USINT, UINT, UDINT, ULINT}
FixedLengthReal ::= CHOICE {REAL, LREAL}
AnyTime ::= CHOICE {ITIME, TIME, FTIME, LTIME}
AnyDate ::= CHOICE {DATE, TIME_OF_DAY, DATE_AND_TIME}
AnyString ::= CHOICE {STRING, STRING2}
FixedLengthBitString ::= CHOICE {SWORD, WORD, DWORD, LWORD}
BOOL ::= [PRIVATE 1] IMPLICIT BOOLEAN
SINT ::= [PRIVATE 2] IMPLICIT OCTET STRING -- 1 octet
INT ::= [PRIVATE 3] IMPLICIT OCTET STRING -- 2 octets
DINT ::= [PRIVATE 4] IMPLICIT OCTET STRING -- 4 octets
LINT ::= [PRIVATE 5] IMPLICIT OCTET STRING -- 8 octets
USINT ::= [PRIVATE 6] IMPLICIT OCTET STRING -- 1 octet
UINT ::= [PRIVATE 7] IMPLICIT OCTET STRING -- 2 octets
UDINT ::= [PRIVATE 8] IMPLICIT OCTET STRING -- 4 octets
ULINT ::= [PRIVATE 9] IMPLICIT OCTET STRING -- 8 octets
REAL ::= [PRIVATE 10] IMPLICIT OCTET STRING -- 4 octets
LREAL ::= [PRIVATE 11] IMPLICIT OCTET STRING -- 8 octets
STIME ::= [PRIVATE 12] IMPLICIT DINT
DATE ::= [PRIVATE 13] IMPLICIT UINT
TIME_OF_DAY ::= [PRIVATE 14] IMPLICIT UDINT
DATE_AND_TIME ::= [PRIVATE 15] IMPLICIT SEQUENCE {
    time_of_day UDINT,
    date UINT }
STRING ::= [PRIVATE 16] IMPLICIT SEQUENCE {
    charcount      UINT,
    stringcontents OCTET STRING} -- one octet per character
SWORD ::= [PRIVATE 17] IMPLICIT OCTET STRING -- 1 octet
WORD ::= [PRIVATE 18] IMPLICIT OCTET STRING -- 2 octets
DWORD ::= [PRIVATE 19] IMPLICIT OCTET STRING -- 4 octets
LWORD ::= [PRIVATE 20] IMPLICIT OCTET STRING -- 8 octets
STRING2 ::= [PRIVATE 21] IMPLICIT SEQUENCE {
    charcount      UINT,
    string2contents OCTET STRING} -- 2 octets/ character
FTIME ::= [PRIVATE 22] IMPLICIT DINT
LTIME ::= [PRIVATE 23] IMPLICIT LINT
ITIME ::= [PRIVATE 24] IMPLICIT INT
STRINGN ::= [PRIVATE 25] IMPLICIT SEQUENCE {
    charsize      UINT,
    charcount     UINT,
    stringNcontents OCTET STRING} -- N octets/ character
SHORT_STRING ::= [PRIVATE 26] IMPLICIT SEQUENCE {
    charcount     USINT,
    stringcontents OCTET STRING} -- one octet per character
TIME            ::= [PRIVATE 27] IMPLICIT DINT
EPATH           ::= [PRIVATE 28] IMPLICIT OCTET STRING -- IEC 61158-6-2

```

```

STRINGI ::= [PRIVATE 30] IMPLICIT SEQUENCE{
    Stringnum    USINT (number of strings)
    array of     STRUCT
    language1    USINT (first character from ISO 639-2/T)
    language2    USINT (second character from ISO 639-2/T)
    language3    USINT (third character from ISO 639-2/T)
    datatype     EPATH limited to the values 0xD0,0xD5,0xD9,and
0xDA)
    charset      UINT      from IANA MIB Printer Codes
(IETF RFC 1759))
    stringcontents CHOICE OF (SHORT_STRING, STRING, STRING2,
or STRINGN) -- based on datatype field

ARRAY ::= SEQUENCE OF NetworkData -- All of same base type
STRUCT ::= SEQUENCE OF NetworkData -- May be different types
FunctionBlockData ::= SET{
    inputs    [0] IMPLICIT STRUCT OPTIONAL,
    outputs   [1] IMPLICIT STRUCT OPTIONAL,
    controlInputs [2] IMPLICIT STRUCT OPTIONAL,
    controlOutputs [3] IMPLICIT STRUCT OPTIONAL,

```

4.2.4 Data type specification / dictionaries

The definition of an object may include text that defines attributes. Attributes shall be assigned a *Data Type* in an object specification. The Data Type may be one of those defined in this standard or may be an object specific extension to this standard. The following definition shall provide a *Type Specification* for data and shall provide a structure for extending or deriving new data types based on existing defined types.

```

Dictionary ::= CHOICE {VariableDictionary, TypeDictionary}
VariableDictionary ::= SEQUENCE OF VariableDictionaryEntry
VariableDictionaryEntry ::= SEQUENCE{
    name AnyString,
    id FixedLengthInteger,
    type TypeID,
    ranges SEQUENCE OF Subrange, -- for arrays
    accessPrivilege BOOL {READ_ONLY(0), READ_WRITE(1)}
TypeID ::= OCTET STRING -- ASN.1 encoded tag value of the
-- DataTypeSpecification module
Subrange ::= SEQUENCE {
    minValue FixedLengthInteger,
    maxValue FixedLengthInteger}
TypeDictionary ::= SEQUENCE OF TypeDictionaryEntry
TypeDictionaryEntry ::= SEQUENCE {
    name AnyString,
    type TypeID,
    spec DataTypeSpecification}

```

```

DataTypeSpecification ::= CHOICE {
    alt      [PRIVATE 0] IMPLICIT AlternateTypeSpec,
    bool     [PRIVATE 1] IMPLICIT NULL, -- BOOL
    sint     [PRIVATE 2] IMPLICIT NULL, -- SINT
    int      [PRIVATE 3] IMPLICIT NULL, -- INT
    dint     [PRIVATE 4] IMPLICIT NULL, -- DINT
    lint     [PRIVATE 5] IMPLICIT NULL, -- LINT
    usint    [PRIVATE 6] IMPLICIT NULL, -- USINT
    uint     [PRIVATE 7] IMPLICIT NULL, -- UINT
    udint    [PRIVATE 8] IMPLICIT NULL, -- UDINT
    ulint    [PRIVATE 9] IMPLICIT NULL, -- ULINT
    real     [PRIVATE 10] IMPLICIT NULL, -- REAL
    lreal    [PRIVATE 11] IMPLICIT NULL, -- LREAL
    stime    [PRIVATE 12] IMPLICIT NULL, -- TIME
    date     [PRIVATE 13] IMPLICIT NULL, -- DATE
    tod      [PRIVATE 14] IMPLICIT NULL, -- TIME_OF_DAY
    dat      [PRIVATE 15] IMPLICIT NULL, -- DATE_AND_TIME
    str1     [PRIVATE 16] IMPLICIT NULL, -- STRING
    sword    [PRIVATE 17] IMPLICIT NULL, -- SWORD
    word     [PRIVATE 18] IMPLICIT NULL, -- WORD
    dword    [PRIVATE 19] IMPLICIT NULL, -- DWORD
    lword    [PRIVATE 20] IMPLICIT NULL, -- LWORD
    str2     [PRIVATE 21] IMPLICIT NULL, -- STRING2
    ftime    [PRIVATE 22] IMPLICIT NULL, -- FTIME
    ltime    [PRIVATE 23] IMPLICIT NULL, -- LTIME
    itime    [PRIVATE 24] IMPLICIT NULL, -- ITIME
    strN     [PRIVATE 25] IMPLICIT NULL, -- STRINGN
    shstr    [PRIVATE 26] IMPLICIT NULL, -- SHORT_STRING
    time     [PRIVATE 27] IMPLICIT NULL, -- TIME
    epath    [PRIVATE 28] IMPLICIT NULL, -- EPATH
    strI     [PRIVATE 30] IMPLICIT NULL, -- STRINGI
    constructedData CHOICE {
        abbrevStruc [0] IMPLICIT AbbreviatedStructTypeSpec,
        abbrevArr  [1] IMPLICIT AbbreviatedArrayTypeSpec,
        frmlStruc  [2] IMPLICIT FormalStructTypeSpec,
        frmlArr    [3] IMPLICIT FormalArrayTypeSpec,
        expBitStr  [4] IMPLICIT ExpandedFixedLenBitStrTypeSpec,
        expStr1    [5] IMPLICIT ExpandedStringTypeSpec,
        expStr2    [6] IMPLICIT ExpandedString2TypeSpec
    }
}

AbbreviatedStructTypeSpec ::= UINT
AbbreviatedArrayTypeSpec ::= DataTypeSpecification
FormalStructTypeSpec ::= SEQUENCE OF DataTypeSpecification
FormalArrayTypeSpec ::= SEQUENCE {
    lowBound [0] IMPLICIT FixedLengthInteger, -- Array Lower Bound
    highBound [1] IMPLICIT FixedLengthInteger, -- Array Upper Bound
    Bound
        dataType    DataTypeSpecification }

ExpandedFixedLenBitStrTypeSpec ::= SEQUENCE {
    bitStrType DataTypeSpecification -- SWORD, WORD, DWORD, or
    LWORD
    bitFields [7] IMPLICIT BitFieldDef}

BitFieldDef ::= SEQUENCE OF {
    bitDef [2] IMPLICIT OCTET STRING} -- Length is always 2
octets.
-- First octet contains starting
-- Bit Position. Trailing octet
-- contains the number of bits.

ExpandedStringTypeSpec ::= UINT-- String Length In Octets

```



```

ExpandedString2TypeSpec ::= UINT-- String Length In Octets
AlternateTypeSpec ::= CHOICE {
    directlyDerivedTypeSpec [0]    IMPLICIT    TypeID,
    subrangeTypeSpec        [1]    IMPLICIT    SubrangeTypeSpec ,
    enumeratedTypeSpec       [2]    IMPLICIT    EnumeratedTypeSpec,
    fbTypeSpec               [3]    IMPLICIT    FBTypeSpec}

SubrangeTypeSpec ::= SEQUENCE{
    baseType    TypeID,      -- NOTE    minValue and maxValue
    minValue    FixedLengthInteger, -- shall be within the range
    maxValue    FixedLengthInteger} -- of baseType values

EnumeratedTypeSpec ::= SEQUENCE OF AnyString

BitNameDefintion ::= SEQUENCE {
    bitName    AnyString,
    bitNumber   USINT}

FBTypeSpec ::= SET{
    inputs          [0] IMPLICIT FbtElementSpec OPTIONAL,
    outputs         [1] IMPLICIT FbtElementTypeSpec OPTIONAL,
    controlInputs   [2] IMPLICIT FbtElementTypeSpec OPTIONAL,
    controlOutputs  [3] IMPLICIT FbtElementTypeSpec OPTIONAL}

FbtElementTypeSpec ::= SEQUENCE OF ElementSpec

ElementSpec ::= SEQUENCE {
    name        AnyString,
    typespec     ElementTypeSpec}

ElementTypeSpec ::= CHOICE {
    [0] IMPLICIT TypeID,
    [1] IMPLICIT SubrangeTypeSpec,
    [2] IMPLICIT EnumeratedTypeSpec,
    [3] IMPLICIT FormalArrayTypeSpec,
    [4] IMPLICIT ExpandedStringTypeSpec,
    [5] IMPLICIT ExpandedString2TypeSpec}

```

The following END statement shall terminate the ASN.1 module opened in 4.1.

END.

4.3 Encapsulation abstract syntax

4.3.1 Encapsulation protocol

4.3.1.1 Common note

NOTE In order to send TPDUs over TCP/IP, an encapsulation protocol is required. This subclause defines the encapsulation protocol requirements. The encapsulation protocol is a generic protocol that may be used for transporting data other than Type 2 TPDUs.

4.3.1.2 Encapsulation messages

All encapsulation messages shall be composed of a fixed-length header of 24 octets followed by an optional data portion. The total encapsulation message length (including header) shall be limited to 65 535 octets. The encapsulation message length shall not override length restrictions imposed by the encapsulated protocol. Its structure shall be as shown in Figure 13, where the top of the diagram indicates the portion of the message sent first on the wire.

NOTE The encapsulation data portion of the message is optional.

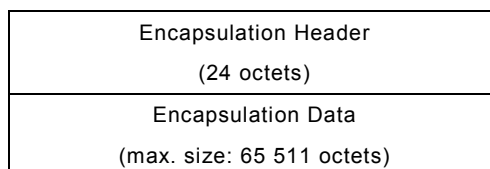


Figure 13 – Encapsulation message

4.3.1.3 Encapsulation header

The encapsulation header shall be as shown in Table 148.

Table 148 – Encapsulation header

Field name	Format	Description
Command	UINT	Encapsulation command
Length	UINT	Length, in octets, of the data portion of the message, i.e., the number of octets following the header
Session handle	UDINT	Session identification (application dependent)
Status	UDINT	Status code
Sender context	ARRAY of 8 USHORT	Information pertinent only to the sender of an encapsulation command
Options	UDINT	Options flags

Multi-octet integer fields in encapsulation messages shall be transmitted as specified in 5.

Although the header contains no explicit information to distinguish between a request and a reply, this information shall be determined in either of two ways:

- implicitly, by the command and the context in which the message is generated. (For example, in the case of the RegisterSession command, the request is generated by an originator and the target generates the reply) ;
- explicitly, by the contents of an encapsulated protocol packet in the data part of the message.

4.3.1.4 Command field

The allocation of command codes shall be as shown in Table 149.

Table 149 – Encapsulation command codes

Command code	Command
0x00	Nop
0x01 – 0x03	Reserved
0x04	ListServices
0x05 – 0x64	Reserved
0x65	ResgisterSession
0x66	UnRegisterSession
0x67 – 0x6E	Reserved
0x6F	SendRRData
0x70	SendUnitData
0x71 – 0xFFFF	Reserved

A device shall accept commands that it does not support without breaking the session or underlying TCP connection. A status code indicating that an unsupported command was received shall be returned to the sender of the message (see 4.3.1.7).

4.3.1.5 Length field

The length field in the header shall specify the size in octets of the data portion of the message. The field shall contain zero for messages that contain no data. The total length of a message shall be the sum of the number contained in the length field plus the 24-octet size of the encapsulation header.

The entire encapsulation message shall be read even if the length may be invalid for a particular command or exceeds the target's internal buffers.

NOTE Failure to read the entire message can result in losing track of the message boundaries in the TCP octet stream.

4.3.1.6 Session handle field

The Session Handle shall be generated by the target and returned to the originator in response to a RegisterSession request. The originator shall insert it in all subsequent session requests (requests using session commands) to that particular target. In the case where the target initiates and sends a command to the originator, the target shall include this field in the request that it sends to the originator.

NOTE Some commands (i.e., Nop) do not require a session handle, even if a session has been established. Whether or not a particular command requires a session is noted in the description of that command.

4.3.1.7 Status field

The value in the Status field shall indicate whether or not the receiver was able to execute the requested encapsulation command. A value of zero in a reply shall indicate successful execution of the command. In all requests issued by the sender, the Status field shall contain zero. If the receiver receives a request with a non-zero Status field, the request shall be ignored and no reply shall be generated.

NOTE This field does not reflect errors that are generated by an encapsulated protocol packet contained within the data portion of the message. For example, an error encountered during an end node's processing of a Set Attributes service.

The status codes shall be as shown in Table 150.

Table 150 – Encapsulation status codes

Status code	Description
0x00	Success
0x01	The sender issued an invalid or unsupported encapsulation command.
0x02	Insufficient memory resources in the receiver to handle the command.
0x03	Poorly formed or incorrect data in the data portion of the encapsulation message.
0x04 – 0x63	Reserved
0x64	An originator used an invalid session handle when sending an encapsulation message to the target.
0x65	The target received a message of invalid length (see 4.3.1.5).
0x66 – 0x68	Reserved
0x69	Unsupported protocol revision.
0x6A – 0xFFFF	Reserved

4.3.1.8 Sender context field

The sender of the command shall assign the value in the Sender Context field of the header. The receiver shall return this value in its reply.

4.3.1.9 Options field

To encapsulate PDUs from the Network and Transport Layer, the Options flags shall be set to zero. The allocation of Option flags shall be as shown in Table 151.

Table 151 – Options flags

Options flags	Definition
Bits 0 – 15	Allocated for compatibility with existing protocols.
Bits 16 – 31	Area for future expansion.

4.3.2 Command descriptions

4.3.2.1 Nop

Either an originator or a target may send a Nop request. No reply shall be generated. The data portion of the request shall be from 0 to 65 511 octets long. The receiver shall ignore any data that is contained in the message.

NOTE A Nop provides a way for either an originator or target to determine if the TCP connection is still open.

The Nop request encapsulation header shall be as shown in Table 152.

Table 152 – Nop request encapsulation header

Field name	Field value
Command	Nop (0)
Length	Length of data portion
Session handle	Ignored
Status	0
Sender context	Ignored
Options	0

4.3.2.2 RegisterSession

An originator shall send a RegisterSession request to a target to initiate a session.

NOTE See 11.8 for detailed information on establishing and maintaining a session.

The RegisterSession request encapsulation header shall be as shown in Table 153.

Table 153 – RegisterSession request encapsulation header

Field name	Field value
Command	RegisterSession (0x65)
Length	4
Session handle	0
Status	0
Sender context	Sender context
Options	0

The parameters in the data portion shall determine the version of the protocol and any session options as shown in Table 154. The protocol version shall be set to 1. The session options shall be set to 0.

Table 154 – RegisterSession request data portion

Field	Type	Description
Protocol version	UINT	Requested protocol version (1)
Options flags	UINT	Session options (0)

Table 155 below shows the organization of the Options flags:

Table 155 – Options flags

Options flags	Definition
Bits 0 – 7	Allocated for compatibility with existing protocols
Bits 8 – 15	Reserved for future expansion

The target shall send a RegisterSession reply to indicate that it has registered the originator. The reply shall have the same format as the request as shown in Table 156.

Table 156 – RegisterSession reply encapsulation header

Field name	Field value
Command	RegisterSession (0x65)
Length	4
Session handle	Handle returned by target
Status	0
Sender context	Context preserved from request
Options	0

The Session Handle field of the header shall contain a target-generated identifier that the originator shall save and insert in the Session Handle field of the header for all subsequent requests to that target. This field shall be valid only if the Status field is zero (0).

The Sender Context field of the header shall contain the same values present in the original sender request. If the originator has been registered with the target, the Status field shall be zero (0). If the target was unable to register, the Status field shall be non-zero.

The data portion of the reply shall have the same format as the request as shown in Table 157.

Table 157 – RegisterSession reply data portion

Field	Type	Description
Protocol Version	UINT	Protocol version supported
Options	UINT	Session options preserved from request

The Protocol Version field shall equal the requested version if the originator was successfully registered. If the target does not support the requested version of the protocol:

- the session shall not be created;
- the Status field shall be set;
- the target shall return the highest supported version in the Protocol Version field.

If all requested options are supported, the Options field shall return the originator's value. This value shall be zero.

4.3.2.3 UnRegisterSession

Either an originator or a target may send this request to terminate the session. The receiver shall initiate a close of the underlying TCP/IP connection when it receives this request. The session shall also be terminated when the transport connection between the originator and target is terminated. The receiver shall perform any other associated cleanup required on its end. There shall be no reply to this command.

The UnregisterSession request format shall be as shown in Table 158.

Table 158 – UnRegisterSession request encapsulation header

Field name	Field value
Command	UnRegisterSession (0x65)
Length	4
Session Handle	Handle from RegisterSession
Status	0
Sender Context	Sender context
Options	0

The Session Handle shall be set to the value obtained by the original RegisterSession reply. The handle shall become invalid after this request has been received and processed by the target.

4.3.2.4 ListServices

The ListServices request shall be sent to determine which encapsulation service classes the target device supports.

NOTE Each service class has a unique type code, and an optional ASCII name.

One service class is defined, with type code 0x100 and name "Communications". This service class shall indicate that the device supports encapsulation of Type 2 PDU's. All devices that support Type 2 PDU encapsulation shall support the ListServices command and Communications service class.

The ListServices request encapsulation header shall be as shown in Table 159.

Table 159 – ListServices request encapsulation header

Field name	Field value
Command	ListServices (0x04)
Length	0
Session handle	Ignored
Status	0
Sender context	Chosen by sender
Options	0

The receiver shall reply with an encapsulation message consisting of the header and data, as shown in Table 160 and Table 161. The data portion of the reply shall provide the information on the services supported.

Table 160 – ListServices reply encapsulation header

Field name	Field value
Command	ListServices (0x04)
Length	0
Session handle	Ignored
Status	0
Sender context	Preserved from request
Options	0

The data portion of the reply shall contain a 2-octet object count followed by an array of objects describing the service(s) provided, as shown in Table 161.

Table 161 – ListServices reply data portion

Field name	Type	Description
Object count	UINT	Number of services objects to follow
Type code	UINT	Service type code
Object length	UINT	Remaining length in octets
Version	UINT	Encapsulation protocol version
Flags	UINT	Capability Flags
Name	ARRAY of 16 USINT	Name of service

The Type Code shall identify the service class as shown in Table 162.

Table 162 – Service type codes

Service type code	Description
0x00 – 0xFF	Reserved
0x100	Communications service
0x101 – 0xFFFF	Reserved

The Version field shall indicate the version of the service supported by the target to help maintain compatibility between applications.

Each service shall have a different set of capability flags. Unused flags shall be set to zero.

The Capability Flags, defined for the Communications service, shall be as shown in Table 163.

Table 163 – Communications capability flags

Flag value	Description
Bits 0 – 4	Reserved
Bit 5	If the device supports Type 2 PDU encapsulation this bit shall be set (= 1) ; otherwise, it shall be clear (= 0)
Bits 6 – 7	Reserved
Bit 8	Supports Class 0 and 1 UDP-based connections
Bits 9 – 15	Reserved

The Name field shall allow up to a 16-octet, NULL-terminated ASCII string for descriptive purposes only. The 16-octet limit shall include the NULL character.

4.3.2.5 ListIdentity

A connection originator may use the ListIdentity request to locate and identify potential targets. This request shall be sent as a broadcast message using UDP and does not require that a session be established.

The ListIdentity request encapsulation header shall be as shown in Table 164.

Table 164 – ListIdentity request encapsulation header

Field name	Field value
Command	ListIdentity (0x63)
Length	0
Session handle	This field is ignored since a session need not be established before sending the ListIdentity request
Status	0
Sender context	0
Options	0

One reply item is defined for this command, Target Identity, with item type code 0x0C. This item shall be supported (returned) by all Type 2 capable devices.

Each receiver of the ListIdentity command shall reply with an encapsulation message consisting of the header and data, as shown in Table 165 and Table 166. The data portion of the message shall provide the information on the targets identity. The reply shall be sent to the IP address from which the broadcast request was received.

Table 165 – ListIdentity reply encapsulation header

Field name	Field value
Command	List Identity (0x63)
Length	0
Session handle	Ignored
Status	0
Sender context	0
Options	0

The data portion of the reply is structured as a Common Packet Format that contains a 2-octet object count followed by an array of objects providing the target identity, as shown in Table 166.

At a minimum, the CFP 2 Identity item shall be returned and has the format as defined in Table 166. Part of this item definition follows the Get Attribute All service response definition of the Identity object (data returned based on instance one of this object), and may adopt new members if and when new members are added to that service response. Unlike most fields in the Common Packet Format, the Socket Address field shall be sent in big endian order.

Table 166 – ListIdentity reply data portion

Field name	Type	Description
Object count	UINT	Number of Identity objects to follow
Type code	UINT	Code indicating object type of CPF 2 Identity (0x0C)
Object length	UINT	Remaining length in octets
Version	UINT	Encapsulation protocol version
Socket Address	ARRAY of :	Socket Address
	INT	sin_family (big-endian)
	UINT	sin_port (big-endian)
	UDINT	sin_addr (big-endian)
	ARRAY of USINT	sin_zero (length of 8) (big-endian)
Vendor ID ^a	UINT	Device manufacturers Vendor ID
Device Type ^a	UINT	Device Type of product
Product Code ^a	UINT	Product Code assigned with respect to device type
Revision ^a	USINT[2]	Device revision
Status ^a	WORD	Current status of device
Serial Number ^a	UDINT	Serial number of device
Product Name ^a	SHORT_STRING	Human readable description of device
State ^a	USINT	Current state of device
^a These parameters are further defined by the corresponding instance attribute of the Identity object.		

4.3.2.6 ListInterfaces

The optional ListInterfaces request shall be used by a connection originator to identify potential non-Type 2 communication interfaces associated with the target. A session need not be established to send this command.

The ListInterfaces request encapsulation header shall be as shown in Table 167.

Table 167 – ListInterfaces request encapsulation header

Field name	Field value
Command	List Interfaces (0x64)
Length	0
Session handle	Ignored
Status	0
Sender context	0
Options	0

If supported, the receiver of a ListInterfaces request command shall reply an encapsulation message consisting of the header and data, as shown below in Table 168.

Table 168 – ListInterfaces reply encapsulation header

Field name	Field value
Command	List Interfaces (0x64)
Length	0
Session handle	Ignored
Status	0
Sender context	0
Options	0

The data portion of the message is structured as a Common Packet Format, which contains a 2-octet object count followed by an array of objects providing information on the non-Type 2 communication interfaces associated with the target.

There are no publicly defined items returned with this reply. The vendor-specific item(s) which is/are returned shall, at a minimum, return a 32 bit Interface Handle which is used by other encapsulation commands, for example, the SendRRData command.

4.3.2.7 SendRRData

A SendRRData request shall transfer an encapsulated request/reply packet between the originator and target, where the originator initiates the command. The actual request/reply packets shall be encapsulated in the data portion of the message and shall be the responsibility of the target and originator. The SendRRData command shall be used to send encapsulated UCMM messages.

The SendRRData request encapsulation header shall be as shown in Table 169.

Table 169 – SendRRData request encapsulation header

Field name	Field value
Command	SendRRData (0x6F)
Length	Length of data portion
Session handle	Handle returned by RegisterSession
Status	0
Sender context	Sender context
Options	0

The data portion of the message shall contain request parameters and the encapsulated protocol packet as shown in Table 170.

Table 170 – SendRRData request data portion

Field name	Type	Description
Interface handle	UDINT	Shall be 0
Timeout	UINT	Operation Timeout
Encapsulated protocol packet (see Common Packet Format specification in 4.3.3)		

The Interface handle shall identify the Communications Interface to which the request is directed. This handle shall be 0 for encapsulating Type 2 PDUs.

The target shall abort the requested operation after the timeout expires. Timeout values shall be expressed in seconds. The minimum timeout value shall be one (1) second; the maximum value shall be 65 535 s (about 18 h). A value of zero shall indicate that the operation shall only be aborted when the encapsulated protocol times-out. Since the Network and Transport Layer already defines a timeout mechanisms for both connected and unconnected messages, the Timeout field shall be zero.

The encapsulated protocol packet shall be encoded in a Common Packet Format as shown in 4.3.3.

The SendRRData reply, as shown in Table 171, shall contain data in response to the SendRRData request. The reply to the original encapsulated protocol request shall be contained in the data portion of the SendRRData reply.

Table 171 – SendRRData reply encapsulation header

Field name	Field value
Command	SendRRData (0x6F)
Length	Length of data portion
Session handle	Handle returned by RegisterSession
Status	0
Sender context	Preserved sender context
Options	0

The format of the data portion of the reply message shall be the same as that of the SendRRData request message.

4.3.2.8 SendUnitData

The SendUnitData request shall send encapsulated connected messages. A reply need not be returned. This command may be used when the encapsulated protocol has its own underlying end-to-end transport mechanism.

The format of the SendUnitData request shall be as shown in Table 172.

Table 172 – SendUnitData request encapsulation header

Field name	Field value
Command	SendUnitData (0x70)
Length	Length of data portion
Session handle	Handle returned by RegisterSession
Status	0
Sender context	Sender context
Options	0

The data portion of the message shall contain request parameters as well as the encapsulated protocol packet as shown in Table 173.

Table 173 – SendUnitData request data portion

Field name	Type	Description
Interface handle	UDINT	Shall be 0
Timeout	UINT	Operation timeout
Encapsulated protocol packet (see Common Packet Format specification in 4.3.3).		

Interface handle and Timeout shall be the same as in the SendRRData request.

4.3.3 Common packet format

4.3.3.1 General

The common packet format shall consist of an item count, followed by an address item, then a data item (in that order) as shown in Table 174. Additional optional items may follow.

NOTE The common packet format defines a standard format for protocol packets that are transported with the encapsulation protocol. The common packet format is a general-purpose mechanism designed to accommodate future packet or address types.

Table 174 – Common packet format

Field name	Type	Description
Item count	UINT	Number of items to follow (shall be at least 2)
Address item	Item Struct (see below)	Addressing information for encapsulated packet
Data item	Item Struct (see below)	The encapsulated data packet
Optional additional items.		

The address and data item structure shall be as shown in Table 175.

Table 175 – Address and data item structure

Field name	Type	Description
Type ID	UINT	Type of item encapsulated
Length	UINT	Length in octets of data to follow
Data	Variable	The data (if length >0)

The address type ID's shall be as shown in Table 176.

Table 176 – Address type ID's

Address type ID	Description
0x00	Null (used for UCMM messages)
0x01 – 0xA0	Reserved
0xA1	Connection-based (used for connected messages)
0xA2 – 0x8001	Reserved
0x8002	Sequenced Address Type
0x8003 – 0xFFFF	Reserved

The data type ID's shall be as shown in Table 177.

Table 177 – Data type ID's

Data type ID	Description
0x00 – 0xB0	Reserved
0xB1	Connected Transport PDU
0xB2	Unconnected message
0xB3 – 0x7FFF	Reserved
0x8000	Sockaddr Info, originator-to-target
0x8001	Sockaddr Info, target-to-originator
0x8002 – 0xFFFF	Reserved

4.3.3.2 Address types

4.3.3.2.1 Null

The null address type shall contain only the type id and the length as shown in Table 178. The length shall be zero. No data shall follow the length. Since the null address type contains no routing information, it shall be used when the protocol packet itself contains any necessary routing information. The null type shall be used for Unconnected Messages.

Table 178 – Null address type

Field name	Type	Field value
Type ID	UINT	0
Length	UINT	0

4.3.3.2.2 Connected

This address type shall be used when the encapsulated protocol is connection-oriented. The data shall contain a connection identifier, as shown in Table 179.

NOTE Connection identifiers are exchanged in the Forward_Open service of the Connection Manager.

Table 179 – Connected address type

Field name	Type	Field value
Type ID	UINT	0xA1
Length	UINT	4
Data	UDINT	Connection Identifier

4.3.3.2.3 Sequenced address type

This address type shall be used for class 0 and class 1 connected data. The data shall contain a connection identifier and a sequence number, as shown in Table 180. Usage is further described in 11.3.3.

Table 180 – Sequenced address type

Field name	Type	Field value
Type ID	UINT	0x8002
Length	UINT	8
Data	UDINT	Connection Identifier
	UDINT	Sequence Number

4.3.3.3 Data types

4.3.3.3.1 Unconnected

The data type that encapsulates an unconnected message shall be as shown in Table 181.

Table 181 – UCMM data type

Field name	Type	Field value
Type ID	UINT	0xB2
Length	UINT	Length, in octets, of the unconnected message
Data	Variable	The unconnected message

The format of the unconnected message shall be the same as PDU destined for the Message Router. The UCMM header, defined in 4.1.3, shall not be present. The context field in the encapsulation header shall be used for unconnected request/reply matching.

4.3.3.3.2 Connected

The data type that encapsulates a connected transport PDU shall be as shown in Table 182.

Table 182 – Connected data type

Field name	Type	Field value
Type ID	UINT	0xB1
Length	UINT	Length, in octets, of the transport PDU
Data	Variable	The transport PDU

4.3.3.3.3 Sockaddr info

The Sockaddr Info items shall be used to encapsulate socket address information necessary to send datagrams (the connected data) between the target and originator. There are separate items for originator-to-target and target-to-originator socket information.

The Sockaddr Info items shall have the structure shown in Table 183.

Table 183 – Sockaddr info items

Field name	Type	Field value
Type ID	UINT	0x8000, 0x8001
Length	UINT	16 (octets)
Data	ARRAY	See below

The data portion of the Sockaddr Info type shall follow the format of the sockaddr_in structure as defined in the Winsock specification, version 1.1. The sockaddr_in structure is shown below:

```
struct sockaddr_in {
    short    sin_family;
    u_short  sin_port;
    struct    in_addr sin_addr;
    char     sin_zero[8];
};
```

The multi-octet fields in the sockaddr_in structure shall be returned in TCP/IP octet order, since this is the octet ordering used by the Winsock API.

5 Transfer syntax

5.1 Compact encoding

5.1.1 Encoding rules

This subclause describes the means by which the data types defined in this standard shall be encoded/transferred across the control network. The abstract syntax definition along with a particular set of encoding rules shall result in the transfer syntax. For application user data, a single set of encoding rules shall be defined (Compact Encoding), resulting in the Compact transfer syntax.

Compact Encoding rules shall start with the encoding rules defined in ISO/IEC 8825-1. Compact Encoding then applies optimization rules, starting with the outer most Service Data Unit (SDU) and progressing to each successive encapsulated SDU. Compact Encoding shall define a more efficient encoding mechanism by reducing the amount of information (overhead) transferred between devices.

The difference between a Compact encoded value and an ASN.1 encoded value shall be the removal of the fields describing the type and length of the information. The TAG and LENGTH components of an ASN.1-encoded value shall not be transmitted on the control network. In addition, the Compact Encoding rules shall indicate that octet ordering rules are the reverse of those seen in ASN.1.

Given the conditions listed in 5.1.2, general rules shall be applied to an ASN.1 encoded value to generate a Compact encoded value. The general rules shall be as follows:

- remove the Identifier Octets;
- remove the "TAG" octets specified by ASN.1;
- remove the Length Octets;
- remove the "LENGTH" octets specified by ASN.1;
- reverse the octet ordering for multiple content octets.

5.1.2 Encoding constraints

The representation of a variable value using Compact Encoding shall be possible with the following restrictions:

- the variable type shall be fixed length and shall have no conditional or optional fields;
- the encoding of a given variable shall be represented with a constant number of octets derived from the type specification of this variable.

5.1.3 Examples

5.1.3.1 BOOL encoding

The BOOLEAN encoding shall be performed on a single OCTET as described in Table 184.

Table 184 – BOOLEAN encoding

If the value is:	Then:
FALSE	bit 0 of the octet is 0 ('00'H)
TRUE	bit 0 of the octet is 1 ('01'H)

A FALSE BOOL shall be represented as shown in Table 185.

Table 185 – Example compact encoding of a BOOL value

Octet number	1 st
BOOL	00

5.1.3.2 SignedInteger encoding

The SignedInteger encoding shall be performed as described in Table 186.

Table 186 – Encoding of SignedInteger values

Octet number	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th
SINT	0LSB							
INT	0LSB	1LSB						
DINT	0LSB	1LSB	2LSB	3LSB				
LINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

NOTE The example in Table 187 illustrates the encoding of a variable of type DINT whose value is 0x12345678.

Table 187 – Example compact encoding of a SignedInteger value

Octet number	1 st	2 nd	3 rd	4 th
DINT	78	56	34	12

5.1.3.3 UnsignedInteger encoding

The UnsignedInteger encoding shall be performed as described in Table 188.

Table 188 – UnsignedInteger values

Octet number	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th
USINT	0LSB							
UINT	0LSB	1LSB						
UDINT	0LSB	1LSB	2LSB	3LSB				
ULINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

NOTE Table 189 illustrates the encoding of a variable of type UDINT whose value is 0xAABBCCDD.

Table 189 – Example compact encoding of an UnsignedInteger

Octet number	1 st	2 nd	3 rd	4 th
UDINT	DD	CC	BB	AA

5.1.3.4 FixedLengthReal encoding

The FixedLengthReal encoding shall be performed as described in Table 190.

Table 190 – FixedLengthReal values

Octet number	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th
REAL	0LSB	1LSB	2LSB	3LSB				
LREAL	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

Table 191 illustrates the encoding of a variable (Float1) whose type is REAL and whose value is Float1: = 10,0.

NOTE 1 The assignment of the value is using the IEC 61131-3 notation. The ASN.1 value is {‘41200000’H} in IEEE format : $1,25 \times 2^3$, exponent is 130 (bias 127), fraction is 25.

Table 191 – Example compact encoding of a REAL value

Octet contents	0LSB	1LSB	2LSB	3LSB
REAL	00	00	20	41

Table 192 illustrates the encoding of a variable (Float2) whose type is LREAL and whose value is Float2: = –100,0.

NOTE 2 The ASN.1 value is {‘C059000000000000’H} in IEEE format : $1,5625 \times 2^6$, exponent is 1 029 (bias 1 023), fraction is 0,5625.

Table 192 – Example compact encoding of a LREAL value

Octet contents	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
LREAL	00	00	00	00	00	00	59	C0

5.1.3.5 Time encodings

Table 193 illustrates the encoding of Real numbers with fixedlengths.

Table 193 – FixedLengthReal values

Octet number	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th
TIME	0LSB	1LSB	2LSB	3LSB				
DATE	0LSB	1LSB						
TIME_OF_DAY	0LSB	1LSB	2LSB	3LSB				
DATE_AND_TIME	0LSB- Time	1LSB- Time	2LSB- Time	3LSB- Time	0LSB- Date	1LSB- Date		
FTIME	0LSB	1LSB	2LSB	3LSB				
LTIME	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

5.1.3.6 String encodings

This subclause gives examples of the Compact Encoding of STRING, STRING2, STRINGN, and SHORT_STRING data values.

NOTE The preferred string type for user supplied string data is STRING2 due to international character string requirements.

Table 194 provides a generic illustration of the encoding of a STRING value.

Table 194 – STRING value

	Contents (charcount)		Contents (string contents)
	0LSB	1LSB	0LSB

1 octet characters

Table 195 provides a generic illustration of the encoding of a STRING2 value.

Table 195 – STRING2 value

	Contents (charcount)		Contents (string2contents)	
	0LSB	1LSB	0LSB	1LSB

2 octet characters

Table 196 provides a generic illustration of the encoding of a STRINGN value.

Table 196 – STRINGN value

	Contents (charsize)		Contents (charcount)		Contents (stringNcontents)	
	0LSB	1LSB	0LSB	1LSB	0LSB	NLSB

N octet characters

Table 197 provides a generic illustration of the encoding of a SHORT_STRING value.

Table 197 – SHORT_STRING value

	Contents (charcount)	Contents (short_string)
	0LSB	0LSB

1 octet characters

Table 198 illustrates the encoding of a string variable whose contents equal "Mill". Encoding examples of all string types are presented. Character coding is specified in ISO/IEC 10646. The hexadecimal equivalent is: {'4D696C6C'H} for 8 bit encoding.

Table 198 encodes "Mill" as a STRING type.

Table 198 – Example compact encoding of a STRING value

	Contents (charcount)		Contents (string contents)			
	04	00	4D	69	6C	6C

Table 199 encodes "Mill" as a STRING2 type.

Table 199 – Example compact encoding of STRING2 value

	Contents (charcount)		Contents (string2 contents)							
STRING2	04	00	4D	00	69	00	6C	00	6C	00

Table 200 encodes "Mill" as a SHORT_STRING type.

Table 200 – SHORT_STRING type

	Contents (charcount)	Contents (short_string contents)			
SHORT_STRING	04	4D	69	6C	6C

5.1.3.7 FixedLengthBitString encoding

This subclause provides examples of the Compact Encoding of SWORD, WORD, DWORD, LWORD data values. Figure 14 illustrates the bit placement rules associated with the Compact Encoding of a FixedLengthBitString.

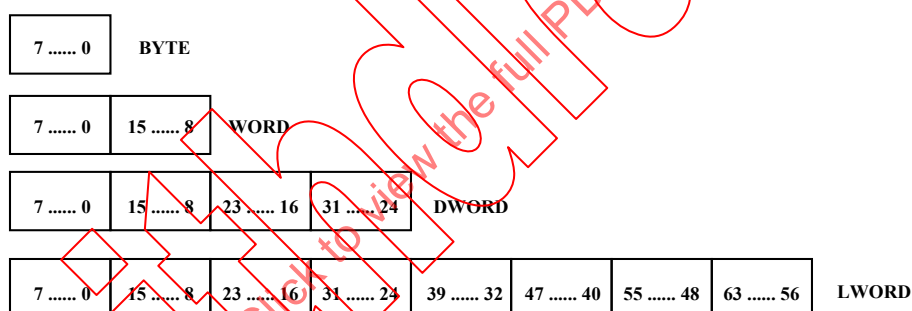


Figure 14 – FixedLengthBitString compact encoding bit placement rules

Figure 15 through Figure 18 illustrate the encoding of SWORD, WORD, DWORD, and LWORD.

Bits In Memory: 7 0
00001111

Compact Encoded BYTE
00001111 or 0x0F

Figure 15 – Example compact encoding of a SWORD FixedLengthBitString

Bits In Memory: 15 0
00001111 11001111

Compact Encoded WORD
11001111 00001111 or 0xCF0F

Figure 16 – Example compact encoding of a WORD FixedLengthBitString

Bits In Memory: 31 0
 00001111 11001111 10110110 00111110

Compact Encoded DWORD
 00111110 10110110 11001111 00001111 or 0x3EB6CF0F

Figure 17 – Example compact encoding of a DWORD FixedLengthBitString

Bits In Memory: 63 0
 11110000 11001111 10110110 00111110 11110000 00101101 00011110 00001111

Compact Encoded LWORD
 00001111 00011110 00101101 11110000 00111110 10110110 11001111 11110000 or 0x0F1E2DF03EB6CF0F

Figure 18 – Example compact encoding of a LWORD FixedLengthBitString

5.1.3.8 Array encoding

5.1.3.8.1 One-dimensional array encoding

The Array encoding shall use the encoding rules for the data types for each element and concatenates the elements which compose the array. The encoded values of the array elements shall be encoded in the same order as they are declared in the corresponding ASN.1 type or variable specification. These elements may be of any data type.

The ASN.1-style definition of a single-dimensional array in the control network shall be:

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,
                        array_dimension_high_bound, NetworkData }
```

Assume the following array definition::

```
ARRAY1 ::= SEQUENCE OF {array_dimension_low_bound := 0,
                        array_dimension_high_bound := 1, UINT}
```

Plugging the UINT values {1,2} into this array definition yields the encoding specified in Table 201.

Table 201 – Example compact encoding of a single dimensional ARRAY

Octet number	1 st	2 nd	3 rd	4 th
ARRAY	01	00	02	00

5.1.3.8.2 Two-dimensional array encoding

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,
                        array_dimension_high_bound,
                        SEQUENCE OF { array_dimension_low_bound,
                        array_dimension_high_bound, NetworkData } }
```

5.1.3.8.3 Three-dimensional array encoding

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,
                        array_dimension_high_bound,
                        SEQUENCE OF { array_dimension_low_bound,
                        array_dimension_high_bound,
                        SEQUENCE OF { array_dimension_low_bound,
                        array_dimension_high_bound, NetworkData } } }
```

Since control network data may comprise either ElementaryData or DerivedData, a new type or variable specification may be required before transmitting the values for the ARRAY.

A multi-dimensional array shall be seen on the wire as a single-dimensional array. The order of the array elements shall be maintained via the packing/unpacking sequence followed by the end nodes. The sequence followed shall be to access the Nth dimension of the array for all values of the other dimensions.

This shall be achieved by first accessing the array with all dimensions set to their initial index values. After this the Nth dimension is incremented through all of its index values. When the end of the index range for the Nth dimension is reached, the (N-1)th dimension is incremented, and the Nth dimension is set to its initial index value. This process is repeated until all of the array's dimensions have reached the ends of their index ranges, and results in the array being packed into the message buffer as a single-dimensional array. The same procedure is followed to unpack the single-dimensional array into a multi-dimensional array.

The two-dimensional array

```
ARRAY1 [0..1 , 0..2] of UINT := { { 1, 2, 3 },
                                     { 4, 5, 6 } }
```

results in the data stream shown in Table 202 when it is packed into a single-dimensional array following the compact encoding rules:

Table 202 – Example compact encoding of a multi-dimensional ARRAY

Octet number	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th	8 th	9 th	10 th	11 th	12 th
ARRAY	01	00	02	00	03	00	04	00	05	00	06	00

5.1.3.9 Structure encoding

The structure encoding shall use the encoding rules for the data types for each element and concatenates the elements which compose the structure.

The encoded values of the structure elements shall be encoded in the same order as they are declared in the corresponding ASN.1 type or variable specification. These elements may be of any Data type.

STRUCT ::= SEQUENCE OF NetworkData -- May be different types

Since NetworkData may be comprised of either ElementaryData or DerivedData, a new type or variable specification may be required before transmitting the values for the STRUCT.

Assume the following structure definition:

newStruct ::= SEQUENCE { BOOL,UINT,DINT}

Plugging the values {TRUE,'1234'H,'56789ABC'H} into the structure results in the encoding as specified in Table 203.

Table 203 – Example compact encoding of a STRUCTURE

Octet number	1 st	2 nd	3 rd	4 th	5 th	6 th	7 th
newStruct	01	34	12	BC	9A	78	56

5.1.3.10 Complex encoded data format examples

5.1.3.10.1 General

The examples 5.1.3.10.2 and 5.1.3.10.3 show how more complex data formats shall be packed. Example 5.1.3.10.2 shows the packing of an array of structures. Example 5.1.3.10.3 shows how a structure with an array element is packed.

5.1.3.10.2 Example 1: encoding an array of structures:

```
STRUCT1 ::= SEQUENCE OF {
    UINT      ele1;
    USINT     ele2;
    USINT     ele3;
    USINT     ele4;
    UINT      ele5
}

ARRAY1 [ 0..1 , 0..2 ] of STRUCT1 := {
    { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 },
      { 11, 12, 13, 14, 15 } },
    { { 15, 14, 13, 12, 11 }, { 10, 9, 8, 7, 6 },
      { 5, 4, 3, 2, 1 } } }
```

results in the following data stream:

```
[01][00][02][03][04][05][00] [06][00][07][08][09][0A][00]
[0B][00][0C][0D][0E][0F][00] [0F][00][0E][0D][0C][0B][00]
[0A][00][09][08][07][06][00] [05][00][04][03][02][01][00]
```

5.1.3.10.3 Example 2: encoding a structure with an array element

```
STRUCT2 ::= SEQUENCE OF {
    UINT      ele1;
    ARRAY [ 0..2 ] of USINT array2;
    UINT      ele5;
}

STRUCT2 := { 1, { 2, 3, 4 }, 5 }
```

results in the following data stream.

```
[01][00] [02][03][04] [05][00]
```

5.2 Data type reporting

5.2.1 Object data representation

Objects may choose to implement a mechanism for reporting Data Type of a particular Attribute or transmitting type information along with the actual data. This subclause defines the means by which Data Typing information is conveyed.

The specification of Data Type information utilizes the ASN.1 methodology specified in the ISO/IEC 8824 series and ISO/IEC 8825-1 with the control network defined optimizations to encode the **DataTypeSpecification** production defined in 4.2.4.

The control network defined optimization is that the Length Octet of a NULL type is not encoded.

NOTE For example, the encoding of 'abc [PRIVATE 1] IMPLICIT NULL' would be: 0xC1 (a tag with no length octet).

5.2.2 Elementary data type reporting

Elementary data types shall be identified using the identification codes defined in Table 204. These codes shall define the encoding of the primitive members of the **DataTypeSpecification**

production. The control network specifies that ASN.1 NULL types shall not report the Length Octet of zero (0).

Table 204 – Identification codes and descriptions of elementary data types

Data type name	Data type code (in hex)	Data type description
BOOL	C1	Logical Boolean with values TRUE and FALSE
SINT	C2	Signed 8-bit integer value
INT	C3	Signed 16-bit integer value
DINT	C4	Signed 32-bit integer value
LINT	C5	Signed 64-bit integer value
USINT	C6	Unsigned 8-bit integer value
UINT	C7	Unsigned 16-bit integer value
UDINT	C8	Unsigned 32-bit integer value
ULINT	C9	Unsigned 64-bit integer value
REAL	CA	32-bit floating point value
LREAL	CB	64-bit floating point value
STIME	CC	Synchronous time information
DATE	CD	Date information
TIME_OF_DAY	CE	Time of day
DATE_AND_TIME	CF	Date and time of day
STRING	D0	character string (1 octet per character)
SWORD	D1	bit string - 8-bits
WORD	D2	bit string - 16-bits
DWORD	D3	bit string - 32-bits
LWORD	D4	bit string - 64-bits
STRING2	D5	character string (2 octets per character)
FTIME	D6	Duration (high resolution)
LTIME	D7	Duration (long)
ITIME	D8	Duration (short)
STRINGN	D9	character string (N octets per character)
SHORT_STRING	DA	character sting (1 octet per character, 1 octet length indicator)
TIME	DB	Duration (milliseconds)
EPATH	DC	Path segments
STRINGI	DE	International Character String

5.2.3 Constructed data type reporting

5.2.3.1 Structure type definition

5.2.3.1.1 General

The control network defines two different methods for reporting Structure type definitions:

- Formal Encoding (FormalStrucTypeSpec);
- Abbreviated Encoding (AbbreviatedStrucTypeSpec).

Formal encoding shall be used to provide a detailed report of the complete structure definition, including the complete definition of all component data types. Abbreviated encoding shall be used to specify a *shorter* form of the structure definition. This *shorter* form shall not include the data types associated with the structure's components.

5.2.3.1.2 Formal encoding for structure type information

The examples 5.2.3.1.3 and 5.2.3.1.4 illustrate formal encoding for structure type specifications. This is actually an example of the encoding of the FormalStrucTypeSpec production defined in 4.2.4.

5.2.3.1.3 Example 1

Table 205 illustrates the encoding of the following structure definition.

```
STRUCT ::= SEQUENCE OF { BOOL, UINT, DINT }
```

Table 205 – Example 1 of formal encoding of a structure type specification

STRUCT type	Type length	Component types	Component types	Component types
		BOOL	UINT	DINT
A2	03	C1	C7	C4

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.3.1.4 Example 2

Figure 19 illustrates the encoding of the following structure definition.

```
STRUCT_MAIN ::= SEQUENCE OF { UINT, STRUCT_SUB, INT }
```

with subelement STRUCT_SUB defined as:

```
STRUCT_SUB ::= SEQUENCE OF { UINT, SINT, INT }
```

Struct Type	Type Length	Component Types						
		UINT	Struct Type	Type Length	Nested Structure Component Types			
					UINT	SINT	INT	INT
A2	07	C7	A2	03	C7	C2	C3	C3

Figure 19 – Example 2 of formal encoding of a structure type specification

5.2.3.1.5 Abbreviated encoding for structure type information

The example 5.2.3.1.6 illustrates the abbreviated encoding for structure type specifications. This is actually an example of the encoding of the AbbreviatedStrucTypeSpec production defined in 4.2.4.

The UINT defined within the AbbreviatedStrucTypeSpec production shall be initialized with a 16 bit Cyclic Redundancy Check (CRC) value (see IEC 61158-4-2:2010). This can be used by application logic to determine whether or not the format of the structure has changed. The octet stream used to produce the CRC is the actual formally encoded (FormalStrucTypeSpec) structure type specification.

5.2.3.1.6 Example

Figure 20 shows the abbreviated encoding of the structure definition presented in 5.2.3.1.5:

Struct Type	Type Length	UINT containing CRC	
A0	02	C7	26

Figure 20 – Example of abbreviated encoding of a structure type specification

NOTE The algorithms presented in this STANDARD are used to generate the CRC value of 0x26C7 from the Formally Encoded type specification: [A2][07][C7][A2][03] [C7][C2][C3][C3].

5.2.3.2 Array type definition

Two different methods for reporting array type definitions shall be:

- Formal Encoding (FormalArrayTypeSpec);
- Abbreviated Encoding (AbbreviatedArrayTypeSpec).

Formal encoding shall be used to provide a detailed report of the complete array definition, including the data content and the array's dimensions. Abbreviated encoding shall be used to specify a *shorter* form of the array definition. This *shorter* form shall not include information specifying the array's dimensions.

5.2.3.3 Formal encoding for array type information

5.2.3.3.1 Common notes

NOTE 1 This subclause contains no normative requirements.

NOTE 2 The examples 5.2.3.3.2 and 5.2.3.3.3 illustrate formal encoding for structure type specifications. This is actually an example of the encoding of the FormalArrayTypeSpec production defined in 4.2.4.

5.2.3.3.2 Example 1

Figure 21 shows the formal encoding of the following array definition

```
ARRAY1 ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 9,
                        UINT
```

Array Type	Type Length	Lower Bound Tag	Lower Bound Length	Lower Bound	Upper Bound Tag	Upper Bound Length	Upper Bound	UINT
A3	07	80	01	00	81	01	09	C7

Figure 21 – Example 1 of formal encoding of an array type specification

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.3.3.3 Example 2

Figure 22 shows the encoding of the following array definition.

```
ARRAY1 ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 19,
                        SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 255,
                        STRUCT_ELE } }
```

in which STRUCT_ELE is defined as:

```
STRUCT_ELE ::= SEQUENCE OF { UINT, SINT, INT }
```

Formal Encoding: [A3][13][80][01][00][81][01][13][A3][0B]80[01][00][81][01][FF][A2][03][C7][C2][C3]

Description:

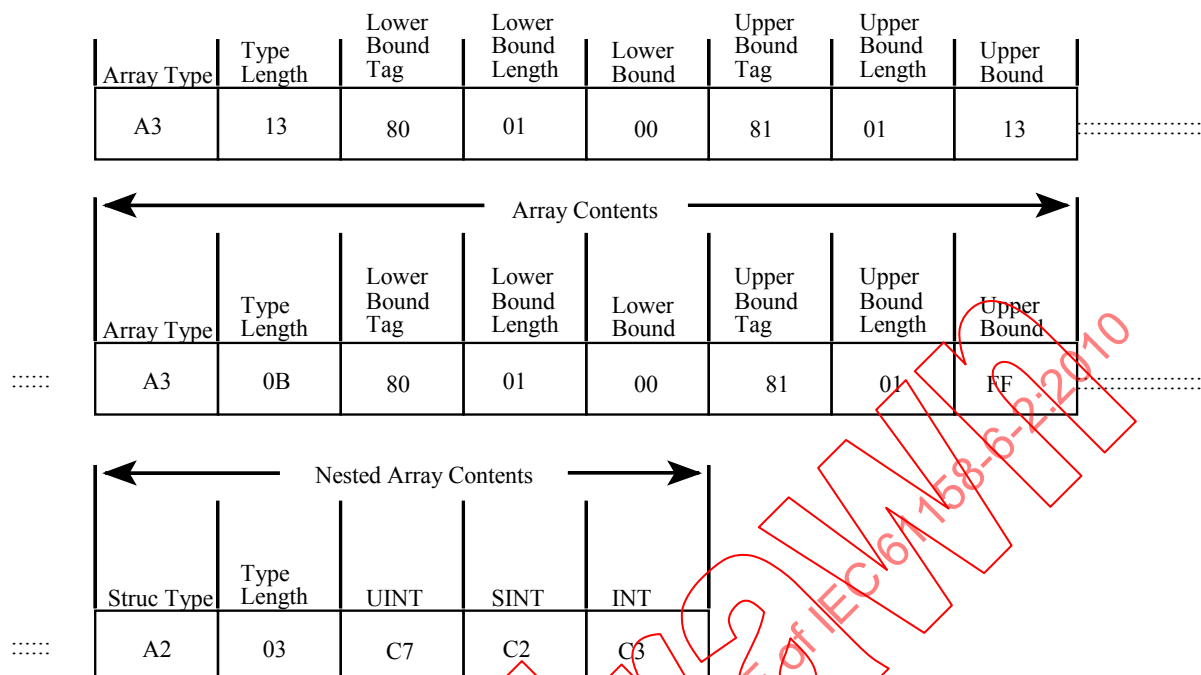


Figure 22 – Example 2 of formal encoding of an array type specification

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.3.4 Abbreviated tag encoding for array type information

5.2.3.4.1 Common notes

NOTE 1 This subclause contains no normative references.

NOTE 2 The abbreviated encoding of an Array type shall not include the information specifying the Array's dimensions. This is actually an example of the encoding of the AbbreviatedArrayTypeSpec production defined in 4.2.4.

5.2.3.4.2 Example 1

Figure 23 shows the abbreviated encoding of the following array definition:

```
ARRAY2 ::= SEQUENCE OF { array_dimension_low_bound := 0,
                          array_dimension_high_bound := 9,
                          UINT }
```

Array Type	Type Length	UINT
A1	01	C7

Figure 23 – Example 1 of abbreviated encoding of an array type specification

NOTE The IMPLICIT NULL types from the DataTypeSpecification production are not followed by a Length Octet of zero (0).

5.2.3.4.3 Example 2

Figure 24 shows the abbreviated encoding of the following array definition:

```

ARRAY ::= SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 19,
                        SEQUENCE OF { array_dimension_low_bound := 0,
                        array_dimension_high_bound := 899,
STRUCT_ELE } }

```

in which STRUCT_ELE is defined as:

```

STRUCT_ELE ::= SEQUENCE OF { UINT, SINT, INT }

```

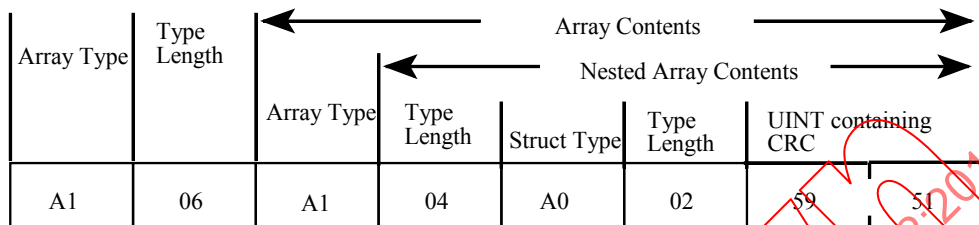


Figure 24 – Example 2 of abbreviated encoding of an array type specification

6 Structure of FAL protocol state machines

Interface to FAL services and protocol machines are specified in this subclause. Conventions used for the descriptions are given in the generic part of this IEC Standard.

This fieldbus follows the structure outlined for Type 1 fieldbus with the following specific features :

- There is no formal definition of AP-Context Machine
- There is a formally-defined FSPM Machine serving as an interface between FAL User and ARPM.
- There are ARPM Machines of two different types :
 - 1) one ARPM machine for connection-less application relationships
 - 2) seven ARPM machines for connection-oriented application relationships
- DMPM Machine is defined at the interface to the Type 2 Data-link layer

7 AP-Context state machine

7.1 Overview

Type 2 supports the AP-Context State Machine specified in, when the Connection object is also supported.

7.2 Connection object state machine

7.2.1 I/O Connection instance behavior

Figure 25 provides a general overview of the behavior associated with an I/O Connection object (instance_type attribute = I/O).

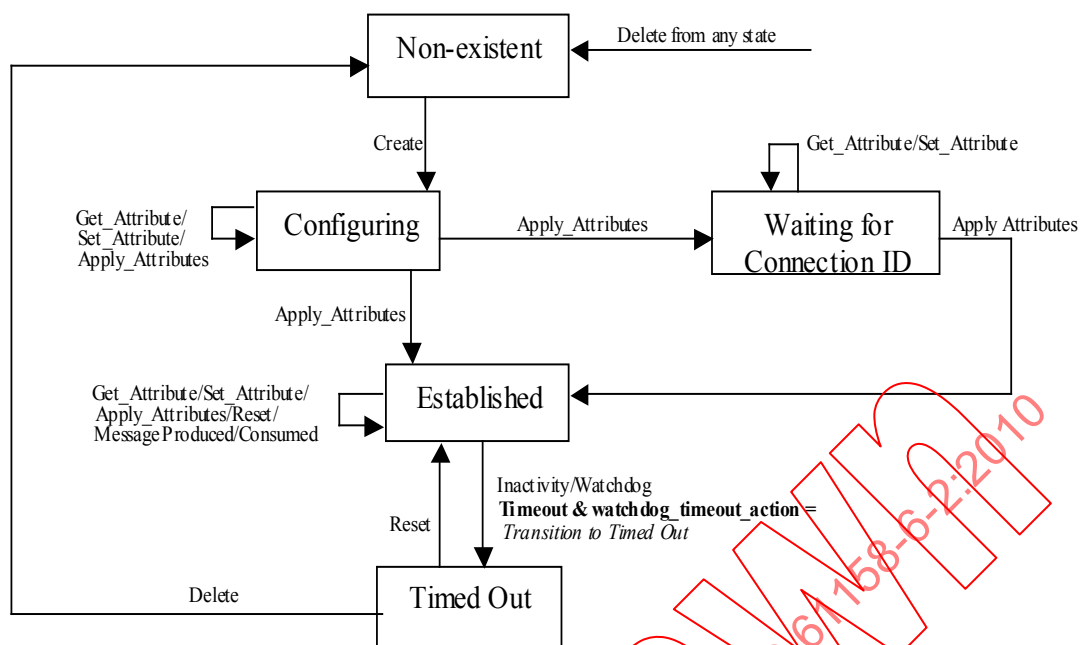


Figure 25 – I/O Connection object state transition diagram

Important: Table 206 provides a detailed State Event Matrix for an I/O Connection object and implementations should be based on this information. This State Event Matrix does not dictate rules with regards to product specific internal logic. Any attempt to access the Connection Class or a Connection object Instance may need to pass through product specific verification. This may result in an error scenario that is not indicated by the SEM in Table 206. This may also result in additional, product specific indications delivered from a Connection object to the application and/or a specific application object. The point to remember is that the Connection object shall exhibit the externally visible behavior specified by the SEM and the attribute definitions.

Table 206 – I/O Connection state event matrix

Event	I/O Connection object state				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
Connection Class receives a Create Request	Class instantiates a Connection object. Set instance_type to I/O. Set all other attributes to default values. Transition to Configuring	Not applicable	Not applicable	Not applicable	Not applicable
Connection Class receives a Delete Request	Error: Object does not exist (General Error Code 16 _{hex})	Release all associated resources. Transition to Non-existent	Release all associated resources. Transition to Non-existent.	Release all associated resources. Transition to Non-existent.	Release all associated resources. Transition to Non-existent.

Event	I/O Connection object state				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
Set_Attribute_Single	Error: Object does not exist (General Error Code 16 _{hex})	Validate/service the request based on internal logic and per the Access Return response.	If request to modify produced or consumed_connection_id then validate the value and service the request. Return appropriate response. If this is a request to access an attribute other than the produced or consumed_connection_id, then return an Error Response whose General Error Code is set to 0C _{hex} (The object can not perform the requested service in its current mode/state)	Validate/service the request based on internal logic and per the Access Rules Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules Return appropriate response.
Get_Attribute_Single	Error: Object does not exist (General Error Code 16 _{hex})	Validate/service the request based on internal logic and per the Access Rules Return response.	Validate/service the request based on internal logic and per the Access Rules Return response.	Validate/service the request based on internal logic and per the Access Rules Return response.	Validate/service the request based on internal logic and per the Access Rules Return response.
Reset	Error: Object does not exist (General Error Code 16 _{hex})	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value =)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0C _{hex})	Cancel the current Inactivity/Watchdog Timer. Using the value in the expected_packet_rate attribute, re-start the Inactivity/Watchdog Timer. A success response is returned even if an Inactivity/Watchdog Timer is not utilized by the Connection object (Client Transport Class 0, expected_packet_rate = 00C _{hex}).	Using the value in the expected_packet_rate attribute, start the Inactivity/Watchdog timer and transition <u>back to the Established state</u> . If the expected_packet_rate attribute has been set to zero (0) while the Connection was in the Timed Out state, then just transition back to Established without activating an Inactivity/Watchdog Timer.

Event	I/O Connection object state				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
Apply_Attributes	Error: Object does not exist (General Error Code 16 _{hex})	Deliver Connection object to the application which validates the attribute information. If either of the connection_id attributes (produced or consumed) needs to be configured and cannot be generated by this module, then perform all the other steps necessary to configure the connection, return a Successful Response and transition to the Waiting For Connection ID state. The inability to generate a produced or consumed connection ID value IS NOT reported as an error. If all attributes are validly configured, then perform all the steps necessary to satisfy this connection, start all required timers, and transition to the Established state. If an error is detected, then an Error Response is returned and the Connection remains in the Configuring state ^a and, if a Client Connection with a production trigger value of 0 or 1 (Cyclic or Change of State), produce initial data.	If either of the connection_id attributes (produced or consumed) still needs to be configured and cannot be generated by this module, then return a Successful Response and remain in the Waiting For Connection ID state. The inability to generate a produced or consumed connection ID value IS NOT reported as an error. If the produced and/or consumed_connection_id attributes are now validly configured, then transition to the Established state and return a Successful Response ¹ and, if a Client Connection with a production trigger value of 0 or 1 (Cyclic or Change of State), produce initial data.	All modifications take place immediately once the Connection has transitioned to the Established state. Return Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0C _{hex})	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0C _{hex})
Receive_Data	Not applicable	Discard the message	Discard the message	If a complete, valid ² message has been received, reset the Inactivity/Watchdog Timer ³ and deliver the I/O Message to the Application. <u>A Connection object shall exhibit the externally visible behavior associated with the current state of its attributes (see Access Rules).</u> If this is a fragmented portion of an I/O Message, process as specified by subnet.	Discard the message

Event	I/O Connection object state				
	Non-Existent	Configuring	Waiting for Connection ID	Established	Timed Out
Send_Message	Not applicable	Return internal error - do not send the message	Return internal error - do not send the message	Transmit the complete I/O Message or fragment as required by the subnet. If this is a Client Connection and the expected_packet_rate attribute is non-zero, restart the Transmission Trigger Timer.	Return internal error - do not send the message
Inactivity/Watchdog Timer expires	Not applicable	Not applicable	Not applicable	Examine the watchdog_timeout_action attribute of the Connection and perform the indicated action. If the watchdog_timeout_action attribute indicates that the Connection is to remain in the Established state (<i>Auto Reset</i>), then immediately re-start the Inactivity/Watchdog Timer.	Not applicable
^a If the configuration indicates that a Message ID needs to be allocated and an available Message ID does not exist in the specified Message Group, then an Error Response whose General Error Code indicates <i>Resource Unavailable</i> (02_{hex}) is returned. If a Connection object attribute value passed the range check when it was initially configured but the attribute value conflicts with another piece of information in the node when the Apply request is processed, then an Error Response is returned whose General Error Code is set to <i>Invalid Attribute Value</i> (09_{hex}) and whose Additional Code is set to the Attribute ID of the <i>offending</i> Connection object Attribute ID. ^b The Connection object verifies that the length of the received I/O Message is less than or equal to the consumed_connection_size attribute prior to processing the message. If the length of the received message is less than or equal to the consumed_connection_size attribute, then the I/O Connection object resets the Inactivity/Watchdog Timer, exhibits the externally visible behavior indicated by its attribute settings, and delivers the message to the Application. If the length of the received message is greater than the consumed_connection_size attribute, then the I/O Connection object immediately discards the message and discontinues any subsequent processing. This is the only message content validation performed by an I/O Connection object. <u>Subsequent validation shall be performed by the Application.</u> ^c If a fragmented message is being received, then the Inactivity/Watchdog Timer is not reset until the entire message has been validly received.					
NOTE Attribute Access Rules are specified in IEC 61158-5-2:2010, 6.2.3.2.1.4					

Important: The Receive_Data event is only delivered to an I/O Connection when a message whose Connection Identifier Field matches the consumed_connection_id attribute is received. If a message is received whose Connection Identifier Field does not match any Established Connection object's consumed_connection_id attribute, then the message is discarded. Connection Identifiers are subnet-type specific.

If an implementation detects that it does not support an Explicit Messaging Service indicated in Table 206, then an Error Response specifying *Service Not Supported* (General Error Code 08) is returned.

7.2.2 Bridged Connection instance behavior

Figure 26 provides a general overview of the behavior associated with a **Bridged** Connection object (**instance_type** attribute = Bridged). Bridged connections are used to make connections *offlink*. Both I/O and Explicit Messaging can be accomplished using this

connection type. The Connection Manager object definition provides more details these types of connections.

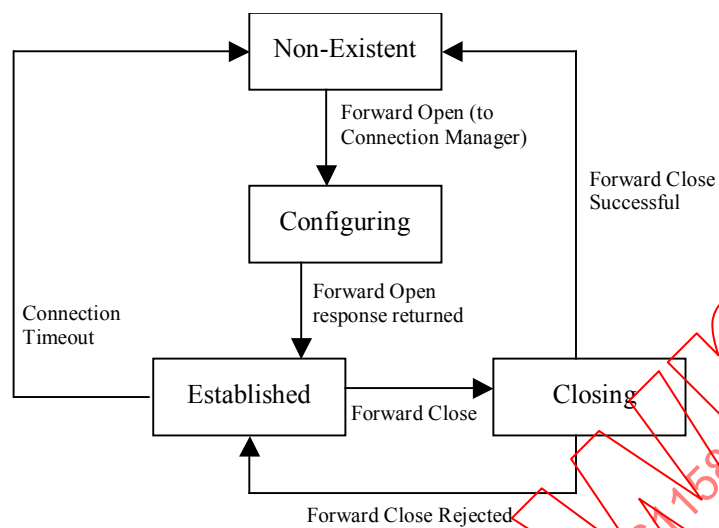


Figure 26 –Bridged Connection object state transition diagram

Table 207 provides a detailed State Event Matrix for a Bridged Connection object.

Table 207 – Bridged Connection state event matrix

Event	Bridged Connection object state			
	Non-Existent	Configuring	Established	Closing
Connection Manager receives a Forward Open Request	Connection Class instantiates a Connection object. Set instance_type to Bridged . Set attributes to value delivered by Forward Open service or default for Bridged connections. Transition to Configuring .	Not applicable	Not applicable	Not applicable
Connection Manager receives notification that connection establishment is complete to target	Not applicable	Transition to Established	Not applicable	Not applicable
Connection Manager receives a Forward Close Request	Not applicable	Ignore event	Transition to Closing	Ignore event
Connection Manager receives a Forward Close Response	Not applicable	Release all resources and transition to Non-Existent	Release all resources and transition to Non-Existent	Release all resources and transition to Non-Existent
Delete	Error: Object does not exist (General Error Code 16 _{hex})	Release all resources and transition to Non-Existent	Release all resources and transition to Non-Existent	Release all resources and transition to Non-Existent
Get_Attribute_Single	Error: Object does not exist (General Error Code 16 _{hex})	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.
Set_Attribute_Single	Error: Object does not exist (General Error Code 16 _{hex})			
Reset	Error: Object does not exist (General Error Code 16 _{hex})	Error: Service Not Supported (General Error Code 08 _{hex})	Error: Service Not Supported (General Error Code 08 _{hex})	Error: Service Not Supported (General Error Code 08 _{hex})
Apply_Attributes	Error: Object does not exist (General Error Code 16 _{hex})	Error: Service Not Supported (General Error Code 08 _{hex})	Error: Service Not Supported (General Error Code 08 _{hex})	Error: Service Not Supported (General Error Code 08 _{hex})
Receive_Data	Not applicable	Ignore event	Invoke send service of Connection object on destination port, passing the received data.	Ignore event
Send_Message	Not applicable	Ignore event	Send data on subnet.	Ignore event
Inactivity/Watchdog Timer expires	Not applicable	Not applicable	Release all resources and transition to Non-Existent	Release all resources and transition to Non-Existent

NOTE Attribute Access Rules are specified in IEC 61158-5-2:2010, 6.2.3.2.1.4

7.2.3 Explicit Messaging Connection instance behavior

Figure 27 provides a general overview of the behavior associated with an **Explicit Messaging Connection object** (**instance_type** attribute = Explicit Messaging).

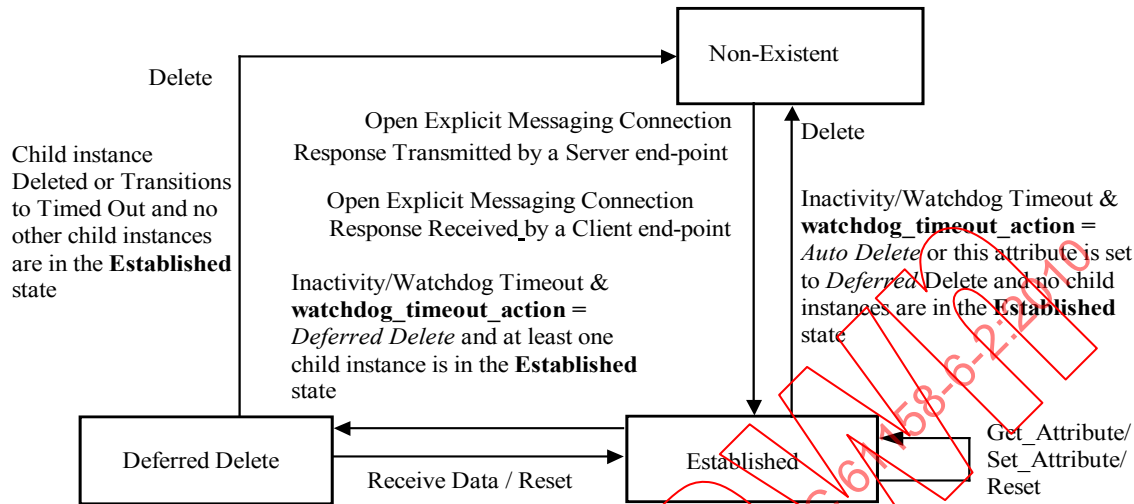


Figure 27 – Explicit Messaging Connection object state transition diagram

Table 208 provides a detailed State Event Matrix for an Explicit Messaging Connection object. Implementations should be based on the information in Table 208.

Table 208 – Explicit Messaging Connection state event matrix

Event	Explicit Messaging Connection object state		
	Non-Existent	Established	Deferred Delete
UCMM receives an Open Explicit Messaging Connection Request/Response and invokes the Create service of the Connection Class	If possible, Class instantiates Connection object. Set instance_type attribute to <i>Explicit Messaging^a</i> . Other attributes are automatically configured using the information in the Open Explicit Messaging Connection Request/Response. Transition to Established . If request received, Transmit Open Explicit Messaging Connection Response.	Not applicable	Not applicable
UCMM receives a Close Request and invokes the Delete service of the Connection Class	Error: Object does not exist (General Error Code 16 _{hex})	Release all associated resources. Transition to Non-existent .	Release all associated resources. Transition to Non-existent .
Connection Class receives a Delete Request	Error: Object does not exist (General Error Code 16 _{hex})	Release all associated resources. Transition to Non-existent .	Release all associated resources. Transition to Non-existent .
Set_Attribute_Single	Error: Object does not exist (General Error Code 16 _{hex})	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.
Get_Attribute_Single	Error: Object does not exist (General Error Code 16 _{hex})	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.	Validate/service the request based on internal logic and per the Access Rules. Return appropriate response.
Reset	Error: Object does not exist (General Error Code 16 _{hex})	Cancel the current Inactivity/Watchdog Timer. Using the value in the expected_packet_rate attribute, re-start the Inactivity/Watchdog Timer.	Using the value in the expected_packet_rate attribute, re-start the Inactivity/Watchdog Timer and transition back to the Established state.
Apply_Attributes	Error: Object does not exist (General Error Code 16 _{hex})	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0Chex)	Error: The object cannot perform the requested service in its current mode/state. (General Error Code value = 0Chex)
Receive_Data	Not applicable	If a valid message or message fragment has been received, then reset the Inactivity/Watchdog Timer ^b . Either process/store the fragment or handle the Explicit Message.	If a valid message or message fragment has been received, then restart the Inactivity/Watchdog Timer ² and transition back to the Established state. Either process/store the fragment or handle the Explicit Message.

Event	Explicit Messaging Connection object state		
	Non-Existent	Established	Deferred Delete
Send_Message	Not applicable	Examine the length of the message to transmit and, if necessary, perform a fragmented series of transmissions. Otherwise, transmit the complete Explicit Message.	Examine the length of the message to transmit and, if necessary, perform a fragmented series of transmissions. Otherwise, transmit the complete Explicit Message.
Inactivity/Watchdog Timer expires	Not applicable	If the watchdog_timeout_action attribute is set to <i>Auto Delete</i> or is set to <i>Deferred Delete</i> and no child connection instances are in the Established state release all associated resources and Transition to Non-existent . If the watchdog_timeout_action attribute is set to <i>Deferred Delete</i> and at least one child connection instance is in the Established state transition to Deferred Delete .	Not applicable.
Child connection instance Deleted or Transitions to Timed Out	Not applicable	Ignore event	If no other child connection instances are in the Established state release all associated resources and transition to Non-existent .
^a If the configuration indicates that a connection resource needs to be allocated and an available resource does not exist, then an Error Response whose General Error Code indicates Resource Unavailable (02hex) is returned. ^b On CP 2/3, the MAC ID field within the Message Header of all Explicit Messages and/or Message Fragments is examined. If the Destination MAC ID is specified in the Connection ID (CAN Identifier Field), then the Source MAC ID of the other end-point shall be specified in the Message Header. If the Source MAC ID is specified in the Connection ID, then the receiving module's MAC ID shall be specified in the Message Header. If either of these checks fail, then the Inactivity/Watchdog Timer IS NOT reset and the message/message fragment is discarded.			
NOTE Attribute Access Rules are specified in IEC 61158-5-2:2010, 6.2.3.2.1.4			

Important: The Receive_Data event is only delivered to an Explicit Messaging Connection when a message whose Connection ID matches the consumed_connection_id attribute is received. If a message is received whose connection id does not match any Established Connection object's consumed_connection_id attribute, then the message is discarded.

If an implementation detects that it does not support an Explicit Messaging Service indicated in Table 208, then an Error Response specifying *Service Not Supported* (General Status Code 08) is returned.

8 FAL service protocol machine (FSPM)

8.1 General

This fieldbus FAL Service Protocol State Machine maps FAL User services onto services of communication objects internal to FAL.

8.2 Primitive definitions

The Objects within FSPM shall provide the following services :

Get_Attribute_All	Reads values of all attributes of the specified object class or instance
Set_Attribute_All	Writes specified values to all attributes of the specified object class or instance
Get_Attribute_List	Reads values of the specified list of attributes of the specified object class or instance
Set_Attribute_List	Writes specified values to the specified list of attributes of the specified object class or instance
Get_Attribute_Single	Reads value of the specified attribute of the specified object class or instance
Set_Attribute_Single	Writes specified value to the specified attribute of the specified object class or instance
Reset	Resets the specified object class or instance
Create	Creates an instance of the specified object class
Delete	Deletes an instance of the specified object class
Start	Starts execution of the specified object
Stop	Stops execution of the specified object
Find_Next_Object_Instance	Finds the identifier of the next unused instance of the specified object class
NOP	Triggers corresponding NOP response from the specified object
Apply_Attributes	Causes pending attribute values to become active in the specified object
Save	Saves attributes contents of the specified object
Restore	Restores attributes contents of the specified object
Group_Sync	Verifies that each member of a group is synchronized to System Time

Refer to IEC 61158-5-2:2010 for detailed definition of these services.

Primitives of Common Services exchanged between UCMM and FAL User are shown in Table 210 and Table 211.

All services have the following common parameters, as shown in Table 209.

Table 209 – Primitives issued by FAL user to FSPM

Common parameters	Req	Ind	Rsp	Cnf
Argument	M			
AREP	M			
Local	S			
UCMM Record identifier	S			
Transport identifier	S			
Receiver/Server Local ID		M		
Path	M	C		
Routing Path	M			
Add. Path	U	U(=)		
Port ID		M		
Result (+)			S	S(=)
AREP				M=
Receiver/Server Local ID			M=	
Service status			M	M(=)
Result (-)			S	S(=)
AREP				M=
Receiver/Server Local ID			M=	
Service status			M	M(=)

Only those argument parameters additional to the common ones are shown in Table 210 and Table 211.

Additional parameters of indication service primitives are the same as those of the corresponding request primitive.

Additional parameters of confirmation service primitives are the same as those of the corresponding response primitive.

Table 210 – Primitives issued by FAL user to FSPM

Primitive name	Source	Additional parameters	Functions
Get_attribute_all.req	FAL User	None	Conveys a request from FAL User to a target object to supply values of all attributes of the specified object class or instance
Set_attribute_all.req	FAL User	Attribute Data	Conveys a request from FAL User to a target object to write specified values of all attributes of the specified object class or instance
Get_attribute_list.req	FAL User	Attribute Count Attribute List	Conveys a request from FAL User to a target object to supply values of specified list of attributes of the specified object class or instance
Set_attribute_list.req	FAL User	Attribute Count Attribute Data	Conveys a request from FAL User to a target object to write specified values of specified list of attributes of the specified object class or instance
Get_attribute_single.req	FAL User	None	Conveys a request from FAL User to a target object to supply values of the specified attribute of the specified object class or instance
Set_attribute_single.req	FAL User	Attribute Data	Conveys a request from FAL User to a target object to write the specified values into specified attribute of the specified object class or instance
Reset.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to reset specified target object class or instance
Create.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to create an instance of the specified object class
Delete.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to delete an instance of the specified object class
Start.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to start an instance of the specified object class
Stop.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to stop an instance of the specified object class
Find_next_object_instance.req	FAL User	Maximum Returned Values	Conveys a request from FAL User to a target device to find the identifier of the next unused instance of the specified object class
NOP.req	FAL User	None	Conveys a request from FAL User to a target object to send back a corresponding NOP response
Apply_Attributes.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to activate pending attribute values
Save.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to save its attributes contents
Restore.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to restore its attributes contents
Group_Sync.req	FAL User	Object Specific Data (optional)	Conveys a request from FAL User to a target object to verify that each member of a group is synchronized to System Time
Get_attribute_all.rsp	FAL User	(+) Attribute Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply values of all attributes of the specified object class or instance
Set_attribute_all.rsp	FAL User	(+) None (-) Specific Status Code	Returns a response from FAL User to a target object to write specified values of all attributes of the specified object class or instance
Get_attribute_list.rsp	FAL User	(+) Attribute Count Attribute Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply values of specified list of attributes of the specified object class or instance
Set_attribute_list.rsp	FAL User	(+) Attribute_count Attribute Status List (-) Specific Status Code	Returns a response from FAL User to a target object to write specified values of specified list of attributes of the specified object class or instance
Get_attribute_single.rsp	FAL User	(+) Attribute Data (-) Specific Status Code	Returns a response from FAL User to a target object to supply values of the specified attribute of the specified object class or instance

Primitive name	Source	Additional parameters	Functions
Set_attribute_single.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to write the specified values into specified attribute of the specified object class or instance
Reset.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has been reset
Create.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has been created
Delete.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has been deleted
Start.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has started
Stop.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that the instance of the specified object class has stopped
Find_next_object_instance.rsp	FAL User	(+) Number Of List Members Instance ID List (-) Specific Status Code	Returns a response from FAL User to a target device to find the identifier of the next unused instance of the specified object class
NOP.rsp	FAL User	None	Returns a response from FAL User to a target object to confirm that the NOP has been received
Apply_Attributes.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that pending attribute values have been activated
Save.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that its attributes contents have been saved
Restore.rsp	FAL User	(+) Object Specific Data (optional) (-) Specific Status Code	Returns a response from FAL User to a target object to confirm that its attributes contents have been restored
Group_Sync.rsp	FAL User	(+) IsSynchronized Object Specific Data (optional) (-) Specific Status Code	Conveys a request from FAL User to a target object to verify that each member of a group is synchronized to System Time

Table 211 – Primitives issued by FSPM to FAL user

Primitive name	Source	Additional parameters	Functions
Get_attribute_all.ind	FSPM	same as in .req primitive	Indicates reception of Get_attribute_all.req
Set_attribute_all.ind	FSPM	same as in .req primitive	Indicates reception of Set_attribute_all.req
Get_attribute_list.ind	FSPM	same as in .req primitive	Indicates reception of Get_attribute_list.req
Set_attribute_list.ind	FSPM	same as in .req primitive	Indicates reception of Set_attribute_list.req
Get_attribute_single.ind	FSPM	same as in .req primitive	Indicates reception of Get_attribute_single.req
Set_attribute_single.ind	FSPM	same as in .req primitive	Indicates reception of Set_attribute_single.req
Reset.ind	FSPM	same as in .req primitive	Indicates reception of Reset.req
Create.ind	FSPM	same as in .req primitive	Indicates reception of Create.req
Delete.ind	FSPM	same as in .req primitive	Indicates reception of Delete.req
Start.ind	FSPM	same as in .req primitive	Indicates reception of Start.req
Stop.ind	FSPM	same as in .req primitive	Indicates reception of Stop.req
Find_next_object_instance.ind	FSPM	same as in .req primitive	Indicates reception of Find_next_object_instance.req
NOP.ind	FSPM	same as in .req primitive	Indicates reception of NOP.req
Apply_Attributes.ind	FSPM	same as in .req primitive	Indicates reception of Apply_Attributes.req
Save.ind	FSPM	same as in .req primitive	Indicates reception of Save.req
Restore.ind	FSPM	same as in .req primitive	Indicates reception of Restore.req
Group_Sync.ind	FSPM	same as in .req primitive	Indicates reception of Group_Sync.req
Get_attribute_all.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_attribute_all.rsp
Set_attribute_all.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_attribute_all.rsp
Get_attribute_list.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_attribute_list.rsp
Set_attribute_list.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_attribute_list.rsp
Get_attribute_single.cnf	FSPM	same as in .rsp primitive	Indicates reception of Get_attribute_single.rsp
Set_attribute_single.cnf	FSPM	same as in .rsp primitive	Indicates reception of Set_attribute_single.rsp
Reset.cnf	FSPM	same as in .rsp primitive	Indicates reception of Reset.rsp
Create.cnf	FSPM	same as in .rsp primitive	Indicates reception of Create.rsp
Delete.cnf	FSPM	same as in .rsp primitive	Indicates reception of Delete.rsp
Start.cnf	FSPM	same as in .rsp primitive	Indicates reception of Start.rsp
Stop.cnf	FSPM	same as in .rsp primitive	Indicates reception of Stop.rsp
Find_next_object_instance.cnf	FSPM	same as in .rsp primitive	Indicates reception of Find_next_object_instance.rsp
NOP.cnf	FSPM	same as in .rsp primitive	Indicates reception of NOP.rsp
Apply_Attributes.cnf	FSPM	same as in .rsp primitive	Indicates reception of Apply_Attributes.rsp
Save.cnf	FSPM	same as in .rsp primitive	Indicates reception of Save.rsp
Restore.cnf	FSPM	same as in .rsp primitive	Indicates reception of Restore.rsp
Group_Sync.cnf	FSPM	same as in .rsp primitive	Indicates reception of Group_Sync.rsp

8.3 Parameters of primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 212.

Table 212 – Parameters used with primitives exchanged between FAL user and FSPM

Parameter name	Description
AREP : Local identifier UCMM Record identifier Transport identifier	Identifies the entity to be used to convey the service. This parameter may use a dedicated identifier associated with a local entity, the identifier of a UCMM transaction record previously created, or the transport identifier returned by the connection establishment process and associated with the selected AR.
Receiver/Server Local ID	Generated by the Message Router ASE of the responding node. Identifies locally this invocation of the service. It is used to associate service responses with indications.
Path : Routing Path Additional path	In the request, it specifies the FAL APO or FAL APO element that is the destination of the service request. In the indication, it contains only those segments beyond the logical class segment from the service request, i.e. the optional additional information for the target APO (Add.Path).
Port ID	Indicates on which port of the device the service indication arrived.
Service status : Status Code (mandatory) Extended Status (conditional)	Provides information on the result of service execution. It is returned in all confirmed service response primitives (+ and -). Status code indicates the type of error (see IEC 61158-5-2:2010 for details). Extended status code gives details of the status (see IEC 61158-5-2:2010 for details).
Attribute Data	1) A stream of information containing all of the attributes. Classes/Objects which support this service shall define the format of this parameter. 2) An array of structures, predefined by the system for the given service
Attribute Count	Number of attribute numbers in the attribute list
Attribute list []	List (array) of the attribute numbers of the attributes to get from the class or object
Attribute Number	Number representing the attribute value
Attribute Status	Status information of attribute
Attribute Value	Sequence of data specific to the attribute of the object or class
Object Specific Data	Class/Instance specific parameters. Class/Instance specification shall specify the format.
Instance ID	Value assigned to identify the newly created object
Maximum Returned Values	Maximum number of Instance ID values to be returned in the response message
Number Of List Members	Number of Instance IDs specified in response message
Instance ID List	Returned Instance ID List. The Instance IDs are returned in 16 bit integer fields.
IsSynchronized	Indicates if object is synchronized to the PTP Time Master

8.4 FSPM state machines

FSPM State Machine has got only one possible state : Running.

9 Application relationship protocol machines (ARPMs)

9.1 General

This fieldbus has Application Relationship Protocol Machines for:

- connection-less application relationships,
- connection-oriented application relationships.

Connection-less relationship is of one type only.

Connection-oriented relationships fit into one of 7 transport classes:

- 0 Null (or Base)
- 1 Duplicate Detection
- 2 Acknowledged
- 3 Verified
- 4 Non-blocking
- 5 Non-blocking, Fragmenting
- 6 Multipoint, Fragmenting

Although similar, each of these transport classes has its own ARPM.

9.2 Connection-less ARPM (UCMM)

9.2.1 General

Functions of the connection-less ARPM are provided by the Unconnected Message Manager (UCMM) object. UCMM shall provide an unconnected request/response message services, limited to a single link that supports multiple outstanding messages. The required number of outstanding messages shall be implementation specific. The UCMM shall be present in all nodes, and shall support at least one outstanding message.

9.2.2 Primitive definitions

The UCMM object shall provide the following services:

UCMM_Create	Creates an instance of transaction record. Puts UCMM into Running state. This service is local, it does not result in a PDU being sent.
UCMM_Delete	Deletes existing transaction record. Puts UCMM into Inactive state. This service is local, it does not result in a PDU being sent. This service is local, it does not result in a PDU being sent.
UCMM_Write	Writes an item of application data into Transport PDU buffer ; this results in sending the data to a specified target.
UCMM_Abort	Aborts existing transaction.

Refer to IEC 61158-5-2:2010 for detailed definition of these services.

Primitives exchanged between UCMM and FSPM are shown in the following Table 213 and Table 214.

Table 213 – Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Functions
UCMM_Create_req	FSPM	fixed tag max retries,	Conveys a request from the FSPM to the ARPM to create a transaction record.
UCMM_Delete_req	FSPM	record	Conveys a request from the FSPM to the ARPM to delete previously established transaction record.
UCMM_Write_req	FSPM	record destination ID UCMM service response timeout transaction priority application data	Conveys a request from the FSPM to the ARPM to send an item of application data using a previously created transaction record.
UCMM_Write_rsp	FSPM	record application data	Conveys a response from the FSPM, via Message Router to the ARPM to send a response to UCMM_Send_req using a previously created transaction record.
UCMM_Abort_req	FSPM	record	Aborts the transaction corresponding to the specified transaction record ; removes the packet from the local DLL if it has not yet been transmitted

Table 214 – Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Functions
UCMM_Create_cnf	ARPM	record service_status	Conveys a confirmation from the ARPM to the FSPM that transaction record has been created.
UCMM_Write_ind	ARPM	record source ID application data	Conveys an indication from the ARPM to the Message Router that data has arrived on the previously created transaction record. The Message Router then passes the indication to the appropriate application object.
UCMM_Write_cnf	ARPM	record service status application data	Conveys a confirmation from the ARPM to the Message Router of the execution of FSPM UCMM_Send_req on a previously created transaction record.

9.2.3 Parameters of primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 215.

Table 215 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter name	Description
record	Identifies unambiguously the instance of the temporary connection along which the FSPM has issued a UCMM_Send request primitive. A means for such identification is not specified in this standard.
application data	Conveys FAL-User data.
fixed tag	Set to value of 0x83 for UCMM and 0x88 for Management UCMM
destination ID	Unambiguous identifier (MAC ID) of the source node of UCMM_write_req.
source ID	Unambiguous identifier (MAC ID) of the destination node of UCMM_write_req.
service	The service parameter shall specify the delivery mechanism to use for this message and shall be one of : Request_With_Acknowledge = 2 (retries until acknowledged) Request_With_No_Response = 4 (no acknowledgement) Request_With_No_ACK = 7 (retries until response) Request_With_No_Retry = 8 (with response) NOTE These values correspond to command codes contained in the UCMM header
timeout	Duration of the transaction in microseconds. If no UCMM_Send_cnf is received before the time-out expires, the transaction is aborted and the send_status parameter shall be TIMEOUT.
retries	Conveys the maximum allowable number of retries.
create_status	Status returned with UCMM_Create_conf = - success - cannot create
send_status	Status returned with UCMM_Send_conf = - success - record_not_created - packet_too_big - no_acknowledge - aborted - timeout

9.2.4 UCMM state machines

9.2.4.1 UCMM client

9.2.4.1.1 States

The defined states and their descriptions of the UCMM Client are listed in Table 216.

Table 216 – UCMM client states

State	Description
Inactive	The record for UCMM transfer does not exist.
Running	The record for UCMM transfer has been created.
Waiting for response	Client sent an item of application data and is waiting for a response.

9.2.4.1.2 State transition diagram

Figure 28 shows the state transition diagram of UCMM client.

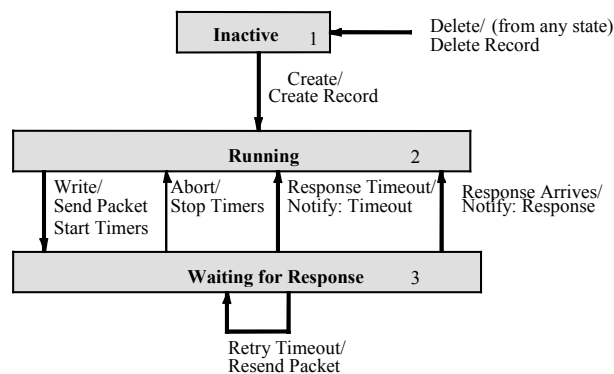


Figure 28 – State transition diagram of UCMM client

9.2.4.1.3 State event matrix

The state transitions for the UCMM Client shall be as specified in Table 217.

Table 217 – State event matrix of UCMM client

Event	State		
	Inactive	Running	Waiting for response
Create.req	1) create record 2) transition to Running		
Delete.req	no action	1) delete record 2) increment SEQUENCE 3) transition to Inactive	1) Delete record 2) increment SEQUENCE 3) transition to Inactive
Write.req	confirm status = INVALID_RECORD	1) send packet 2) start Response Timer 3) initialize Retry Count 4) start Retry Timer 5) transition to Waiting	confirm status = BUSY
Abort.req	confirm status = INVALID_RECORD	no action	1) stop timers 2) increment SEQUENCE 3) transition to Running
Retry Timeout Request packet available			1) Decrement Retry Count IF Retry Count expired 2) Notify: Timeout-No ACK 3) Free request buffer 4) Increment Sequence # 5) Stop Response Timer 6) Transition to Running ELSE. 1) Resend packet 2) Restart Retry Timer
Response Timeout			1) confirm with status = TIMEOUT-No Response 2) increment SEQUENCE 3) transition to Running
Response Arrives		no action	1) confirm with status = SUCCESS 2) Send ACK_RESP, unless response indicates no ACK_RESP desired 3) increment SEQUENCE 4) transition to Running
ACK_REQ Arrives, sequence number matches stored value		No Action	1) Free request buffer 2) stop retry timer
ACK_REQ Arrives, sequence number not equal to stored value		No Action	No Action

The sequence number shall be set to zero at initialization. The sequence number value shall be maintained in the inactive state, to be used when the record transitions to running. The retry timeout value shall be fixed for the link at a value which guarantees that both the client and server nodes have an opportunity to transmit an unscheduled packet.

The response time-out is the response time provided when the record is created.

9.2.4.2 High-end UCMM server

9.2.4.2.1 Functions

When a packet arrives at the server an instance of a transaction shall be created if resources are available. If resources are not available the packet shall be dropped. When the instance is created, a transaction record shall be created for that instance. This record shall be active for the life of this transaction. The transaction shall end:

- when an ACK_RESP is received after a response was sent;
- when the response time timer expires; or
- when a new packet is received after a response was sent.

9.2.4.2.2 States

The defined states and their descriptions of the UCMM High-end Server are listed in Table 218.

Table 218 – High-end UCMM server states

State	Description
Inactive	The record for UCMM transfer does not exist.
Waiting for Response	Data packet arrived with new Transaction ID. New record is created. The Server is waiting for response to be sent by the Server application.
Response sent	Server application sent a response.

9.2.4.2.3 State transitions

Figure 29 shows the high-end UCMM Server state transition diagram.

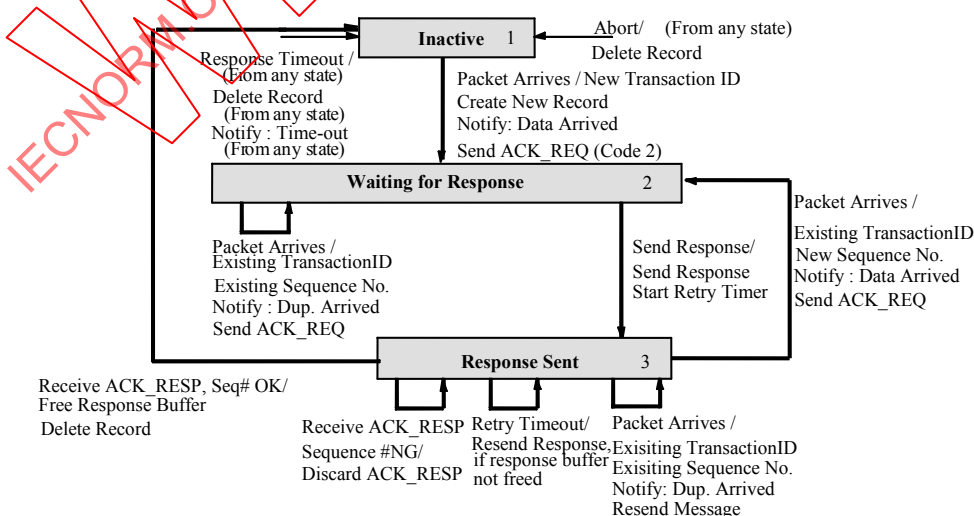


Figure 29 – State transition diagram of high-end UCMM server

9.2.4.2.4 State event matrix

The state transitions for the high-end UCMM server shall be as specified in Table 219.

The sequence number shall be set to zero at initialization. The sequence number value shall be maintained in the Inactive state, to be used when the record transitions to Running. The retry time-out value shall be fixed for the local link.

The response time-out shall be the response time provided when the record is created.

Table 219 – State event matrix of high-end UCMM server

Event	State		
	Inactive	Waiting for response	Response sent
Packet arrives request (code 2) New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) Send ACK_REQ 4) Start Response Timer 5) Transition to Waiting for App Response	Not Applicable	Not Applicable
Packet arrives request (code 7) New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) Start Response Timer 4) Transition to Waiting for App Response	Not Applicable	Not Applicable
Packet arrives Existing Transaction ID and source address New Sequence #	Not Applicable	No Action	1) Update Record 2) Notify: Data Arrived 3) Send ACK_REQ 4) Transition to Waiting for App Response
Packet arrives Existing Transaction ID Existing Sequence #	Not Applicable	1) Notify: Dup. Arrived (Optional) 2) Send ACK_REQ	1) Notify: Dup. Arrived (Optional) 2) Resend Response Message
Response Timer Expires	Not Applicable	1) Notify: Timeout 2) Delete record 3) Transition to Inactive	1) Notify: Timeout 2) Delete record 3) Transition to Inactive
Abort	Error	1) Delete record 2) Transition to Inactive	1) Delete record 2) Transition to Inactive
Send Response	Error	1) Send Response 2) Start Retry Timer 3) Transition to Response Sent	Error
Retry Timeout	Not Applicable	Not Applicable	1) Decrement Retry Count IF Retry Count expired 2) Free Response buffer 3) Stop Response Timer 4) Transition to Inactive ELSE 1) Resend Response message 2) Start Retry Timer
ACK_RESP arrives, sequence number matches stored value	No action	No action	1) Delete record 2) Transition to Inactive
ACK_RESP arrives, sequence number not equal to stored value	No action	No action	No action

The response time is the response time received in the message header.

9.2.4.3 Low-end UCMM server

9.2.4.3.1 Functions

The low-end server cannot support the following features:

- perform response message retries,
- detect duplicate messages.

9.2.4.3.2 States

The defined states and their descriptions of the UCMM Low-end Server are listed in Table 220.

Table 220 – Low-end UCMM server states

State	Description
Inactive	The record for UCMM transfer does not exist.
Waiting for Response	Data packet arrived with new Transaction ID. New record is created. The Server is waiting for response to be sent by the Server application.

9.2.4.3.3 State transitions

Figure 30 shows the low-end UCMM server state transition diagram.

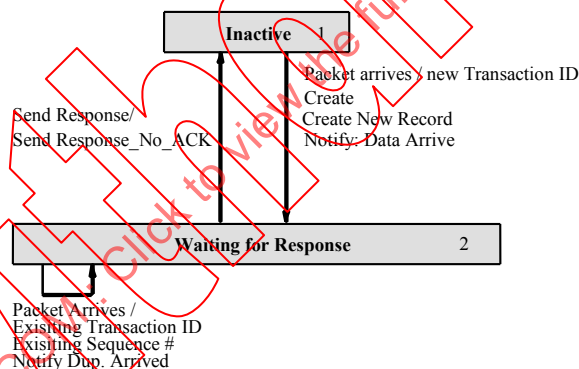


Figure 30 – State transition diagram of low-end UCMM server

9.2.4.3.4 State event matrix

The state transitions for the Low-end UCMM server shall be as specified in Table 221.

Table 221 – State event matrix of low-end UCMM server

Event	State	
	Inactive	Waiting for response
Packet arrives New Transaction ID and source address	1) Create new record 2) Notify: Data Arrived 3) IF Response not immediately available, Send ACK_REQ 4) Transition to Waiting for App Response	Not Applicable
Packet arrives Existing Transaction ID and source address New Sequence #	No action	No action
Packet arrives Existing Transaction ID Existing Sequence #	No action	1) Notify: Dup. Arrived (Optional)
Send Response	Error	1) Send Response_No_ACK 2) Transition to Inactive
ACK_RESP arrives	No action	No Action

9.2.5 Examples of UCMM sequences

Figure 31 shows a sequence diagram for a UCMM with one outstanding message and Figure 32 shows a sequence diagram for a UCMM with multiple outstanding messages.

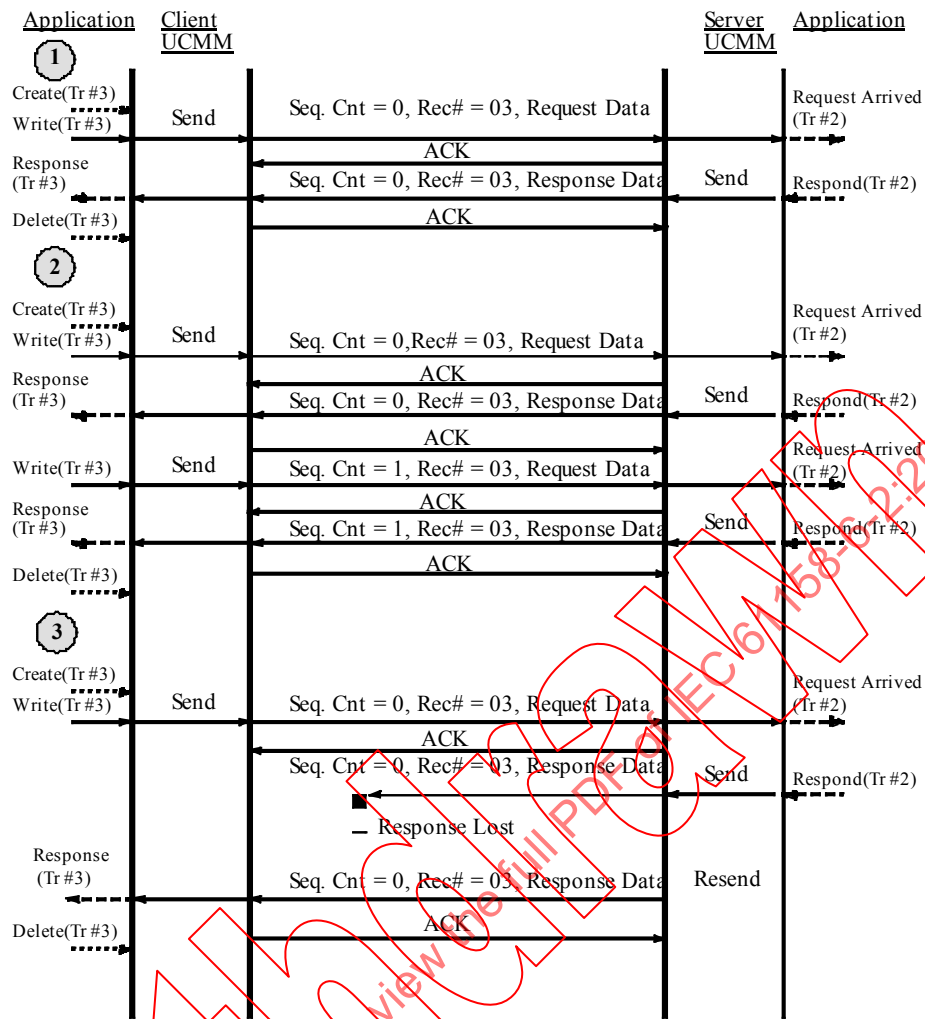


Figure 31 – Sequence diagram for a UCMM with one outstanding message

Example 1. Typical transaction: Instance is created, request is written, response is returned, and instance is deleted.

Example 2. Two transactions/same instance: In this case the instance is opened and two requests are sent.

NOTE 1 The client waits for the first response to return before issuing a second request using the same transaction number.

Example 3. Lost response: In this case the response is lost and the server issues a retry.

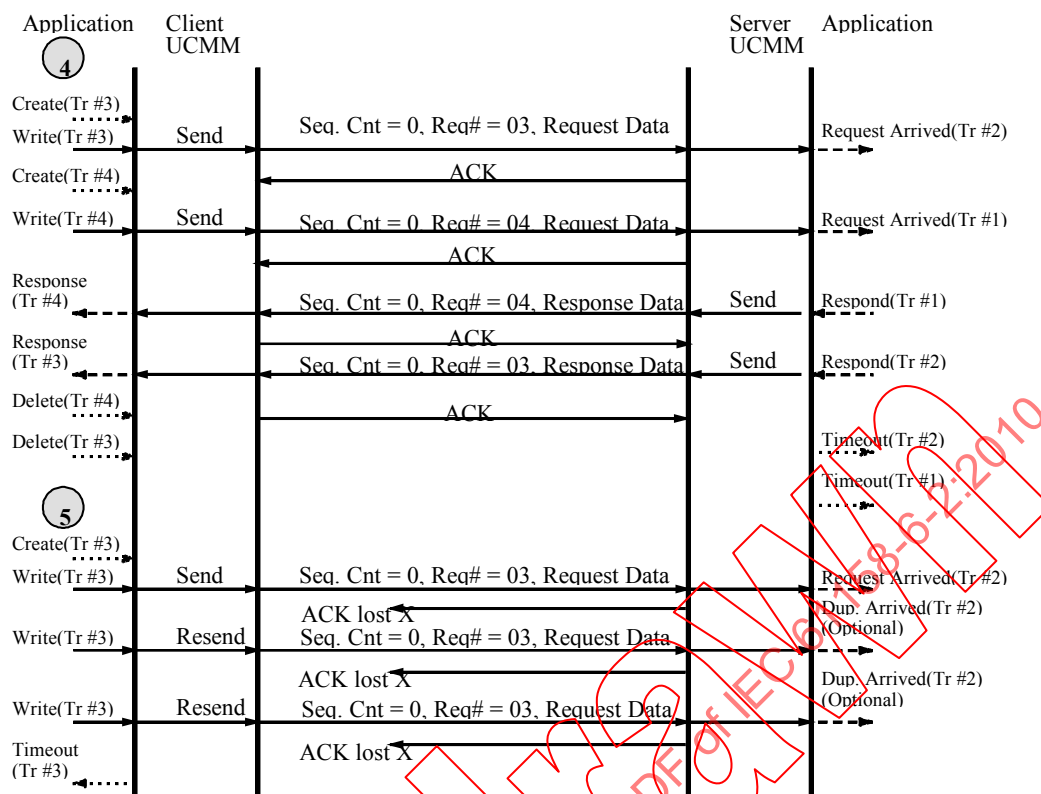


Figure 32 – Sequence diagram for a UCMM with multiple outstanding messages

Example 4. Multiple outstanding: In this case the client creates two instances and sends a second request before the first request has completed.

NOTE 2 The client uses a different transaction number for the second request.

Example 5. Timeout: In this case the server application does not respond and the client issues a timeout. The timeout causes an error status to be returned to the application. Transaction instance shall be deleted.

9.2.6 Management UCMM

Any device that has implemented the Keeper object shall also implement the Management UCMM server. With two exceptions, the Management UCMM server shall be identical to the fixed tag 0x83 UCMM server:

- transactions for the Management UCMM server shall be transmitted on fixed tag 0x88;
- the Management UCMM server shall not transmit any packets unless its node contains a Keeper object in the master state.

Any device may implement the Management UCMM client. The Management UCMM client is identical to the fixed tag 0x83 UCMM client except that transactions for the Management UCMM client shall be transmitted on fixed tag 0x88.

NOTE Messages to the Keeper object are broadcast since all devices are permitted to implement this object. To facilitate communication to the Keeper object without impacting non-Keeper devices, these messages are transmitted on a different fixed tag.

9.3 Connection-oriented ARPMs (transports)

9.3.1 Transport PDU buffer

9.3.1.1 Transport PDU format

The TPDU buffer contains a TPDU packet. The TPDU packet consists of a transport header and data. Applications shall write data to and read data from the TPDU buffers directly. Instances of transports shall write and read the transport header in the TPDU buffer while consumers shall write data to the TPDU buffer and producers shall read data from the TPDU buffer. The TPDU buffers and buffer management are not part of the communication services. This process is shown in Figure 33.

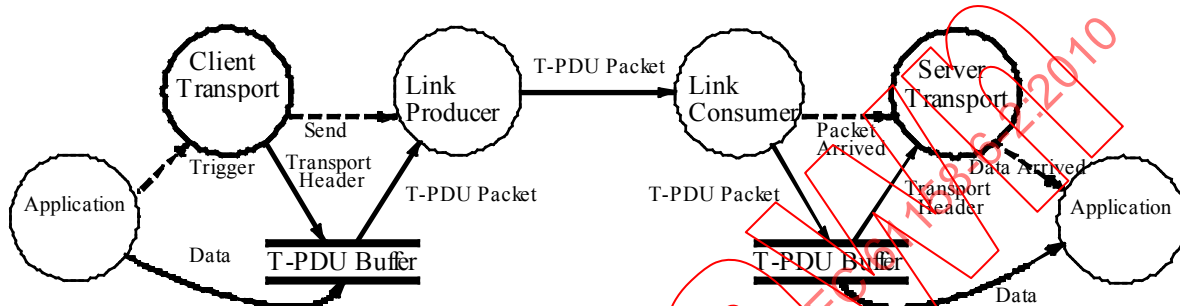


Figure 33 – TPDU buffer

9.3.1.2 Transport PDU buffer management

TPDU buffer management is implementation specific. Implementations that use single buffering, double buffering, overwrite or queuing to manage TPDU buffers are all allowed.

Implementers are responsible for ensuring buffer integrity (atomic access).

Applications are responsible for ensuring that the buffer has been initialized before a trigger is issued. Failure to initialize the buffer shall allow unknown data to be sent. This is true for producers and the client and server sides.

9.3.1.3 Notification

Transport classes can signal the application through events. The valid events are as specified in Table 222.

Table 222 – Notification

Event	Client				Server			
	Class 0	Class 1	Class 2	Class 3	Class 0	Class 1	Class 2	Class 3
Data Arrived					n	n	n	n
Duplicate Arrived						n	n	n
Acknowledgement			n					
Verification				n				

NOTE A *Duplicate Arrived* event is normally not of interest to the application; however, it might be useful in triggering a watchdog timer to indicate that the client's application is alive and triggering.

9.3.2 Transport classes

Applications interface to transport services through the supported transport classes. Table 223 lists the general-purpose transport classes that have been defined for the systems. Each transport class provides a different level of functionality. Transport class 0 provides the

minimum functionality required of a transport class, enabling data transfer between applications.

Table 223 – Transport classes

Class number	Class name
0	Null (or Base)
1	Duplicate Detection
2	Acknowledged
3	Verified
4	Non-blocking
5	Non-blocking, Fragmenting
6	Multipoint, Fragmenting

9.3.3 Common primitive definitions

Transport classes shall provide the following services:

TR_Write	Writes an item of application data into sending Transport PDU buffer, ready to be sent through the pre-established connection.
TR_Trigger	Causes an item of application data or response data previously placed in sending Transport PDU buffer, to be sent through the pre-established connection.
TR_Packet_arrived	Indicates to the Transport instance that a data packet arrived in the receiving Transport PDU buffer. This service is initiated by local action.
TR_Ack_received	Indicates to the user that a data packet arrived in the receiving Transport PDU buffer indicating an acknowledgement of arrival of previously sent data in the receiving Transport PDU buffer. This service is initiated by local action.
TR_Verify	Indicates to the user that a data packet arrived in the receiving Transport PDU buffer indicating a verification of arrival of previously sent data in the receiving user object. This service is initiated by local action.
TR_Status_Update	Indicates to the user that an updated status is present in the Transport PDU buffer.

Refer to IEC 61158-6-2:2010 for detailed definition of these services.

Primitives exchanged between Transport instances and FSPM are shown in the following Table 224 and Table 225.

Table 224 – Primitives issued by FSPM to ARPM

Primitive name	Source	Associated parameters	Functions
TR_Write_req	FSPM	Transport identifier Application data	Conveys a request from the FSPM to the ARPM to write application data into Transport PDU buffer.
TR_Trigger.req	FSPM	Transport identifier	Conveys a request from the FSPM to the ARPM to trigger transfer of application data from Transport PDU buffer to the Link Producer.
TR_Verify.req	FSPM	Transport identifier	Conveys a request from the FSPM to the ARPM to trigger transfer of write verification data into Transport PDU buffer.

Table 225 – Primitives issued by ARPM to FSPM

Primitive name	Source	Associated parameters	Functions
TR_Packet_arrived.ind	ARPM	Transport identifier Data arrived Duplicate arrived	Conveys an indication to the FSPM that an item of data has arrived at the Link Consumer.
TR_Ack_arrived.ind	ARPM	Transport identifier	Conveys an indication to the FSPM that an acknowledgement has arrived at the Link Consumer.
TR_Status_Update.ind	ARPM	Transport identifier	Conveys an indication to the FSPM that a new status has arrived at the Link Consumer.
TR_Verify.cnf	ARPM	Transport identifier	Conveys a confirmation from the ARPM to FSPM to indicate reception of verification DLPDU.

9.3.4 Parameters of common primitives

The parameters used with the primitives exchanged between the FSPM and the ARPM are described in Table 226 .

Table 226 – Parameters used with primitives exchanged between FSPM and ARPM

Parameter name	Description
Transport identifier	Identifier of the instance of the transport invoked in the transaction.
Application data	Contains application data (service request/response parameters or user formatted data).
Data arrived Duplicate arrived	The Data_Arrived and Duplicate_Arrived parameters indicate whether the new packet contains new data, or whether it is just a duplicate from a previously received packet.

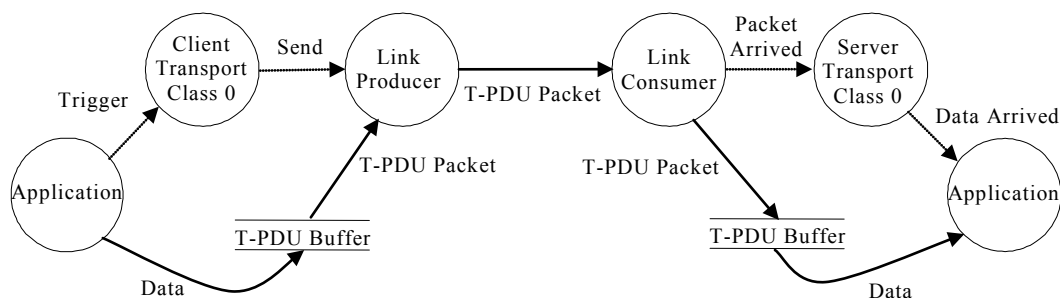
9.3.5 Transport state machines – class 0

9.3.5.1 Functions

Transport class 0 is the simplest transport class, and can use either a point-to-point or multipoint connection. Class 0 is typically used to transmit or receive inputs on a multipoint connection or outputs on a point-to-point connection.

Possible uses of transport class 0 include sampled inputs, standard outputs, cyclic block transfers, diagnostic events, and communication between controllers and operator interface devices.

Since transport class 0 does not detect duplicate data packets, the target application is responsible for detecting them. A duplicate data packet is a retransmission of the last data packet sent. Figure 34 shows the actions which take place during a data transfer using client transport class 0 and server transport class 0.



NOTE The client transport instance does not write a sequence count to the client-side T-PDU buffer, and the server transport instance does not read a sequence count from the server-side T-PDU buffer. The transport header is defined as a don't care

Figure 34 – Data flow diagram using a client transport class 0 and server transport class 0

Figure 35 shows the sequence in which actions take place when transferring data using client transport class 0 and server transport class 0.

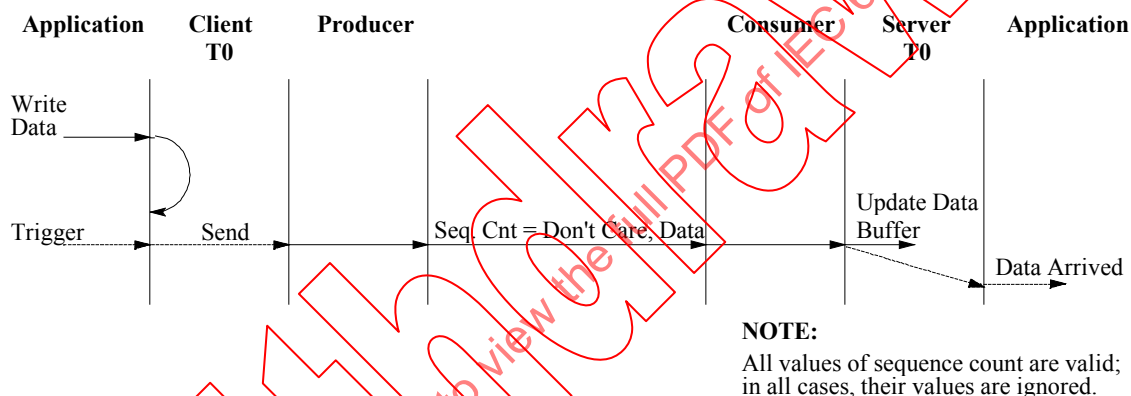


Figure 35 – Sequence diagram of data transfer using transport class 0

9.3.5.2 Transport class 0 client

9.3.5.2.1 Functions

The function of client transport class 0 is just to pass the *trigger* service from the application to the producer as a *send* service. It is useful for the client transport class 0 to be shown from a high-level design view. In actual implementations, the application may trigger the producer directly. An *idle* state is provided for consistency with other transport classes. In the *idle* state all events, except *delete* and *start*, are ignored.

A class 0 client transport is responsible for:

- accepting the client application's trigger that it has written a data packet to the client-side TPDU buffer,
- writing a transport header to the TPDU buffer for each packet that the client application writes to the client-side TPDU buffer (optional),
- triggering the producing node of the network connection to produce the TPDU packet on the link and send it to the consuming node of the network connection.

9.3.5.2.2 States

The defined states and their descriptions of the Class 0 Transport Client are listed in Table 227.

Table 227 – Class 0 transport client states

State	Description
Non-existent	The instance of Transport Class 0 does not exist.
Idle	The instance of Transport Class 0 has been created but not yet started.
Running	The instance of Transport Class 0 has been created and started.

9.3.5.2.3 State transitions

State transition diagram of the class 0 client transport is shown in Figure 36.

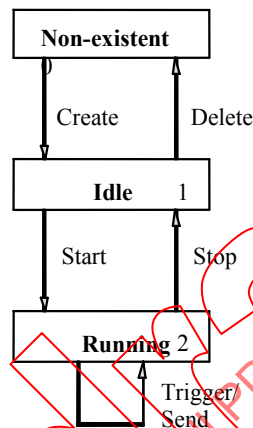


Figure 36 – Class 0 client STD

9.3.5.2.4 State event matrix

State event matrix of the class 0 client transport is shown in Table 228.

Table 228 – Class 0 client SEM

Event	State		
	Non-existent	Idle	Running
Create	Transition to Idle	Error	Error
Delete	Error	Transition to Non-existent	Error
Start	Error	Transition to Running	Error
Stop	Error	No Action	Transition to Idle
Trigger	Error	No Action	Send

9.3.5.3 Transport class 0 server

9.3.5.3.1 Function

The function of server transport class 0 is to pass the *packet arrived* event from the consumer to the producer as a *data arrived* event. It is useful for the server transport class 0 to be shown from a high-level design view. In actual implementations, the application may receive the event directly from the consumer. In the *idle* state all packet arrivals are ignored.

A class 0 server transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side TPDU buffer,

- locating and reading the transport header of each packet that the consuming node of the network connection writes to the server-side TPDU buffer,
- notifying the server application that data has been written to the server-side TPDU buffer.

9.3.5.3.2 States

The defined states and their descriptions of the Class 0 Transport Server are listed in Table 229.

Table 229 – Class 0 transport server states

State	Description
Non-existent	The instance of Transport Class 0 does not exist.
Idle	The instance of Transport Class 0 has been created but not yet started.
Running	The instance of Transport Class 0 has been created and started.

9.3.5.3.3 State transitions

State transitions of the class 0 server transport are shown in Figure 37.

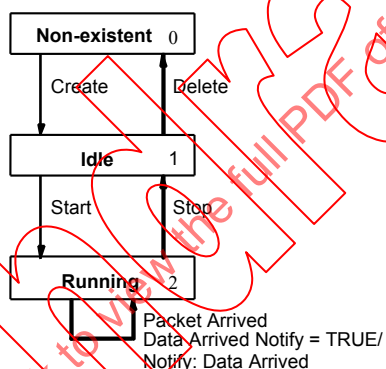


Figure 37 – Class 0 server STD

9.3.5.3.4 State event matrix

State event matrix of the class 0 server transport is shown in Table 230.

Table 230 – Class 0 server SEM

Event	State		
	Non-existent	Idle	Running
Create	Transition to Idle	Error	Error
Delete	Error	Transition to Non-existent	Error
Start	Error	Transition to Running	Error
Stop	Error	No action	Transition to Idle
Packet Arrived & Data Arrived Notify = TRUE	No action	No action	Notify: Data Arrived

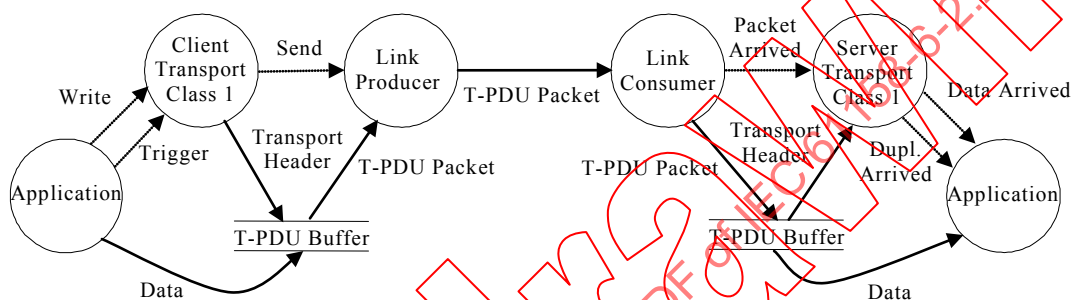
9.3.6 Transport state machines – class 1

9.3.6.1 Functions

Transport class 1, like class 0, shall allow either a point-to-point or multipoint connection. Unlike class 0, which has no transport header, the class 1 transport shall include a sequence number, which is used to detect the delivery of duplicate data packets.

NOTE 1 Class 1 is preferred to class 0 for targets because the target application does not have to perform duplicate detection, which reduces the overhead required in those end nodes that support the change of state transport trigger. Class 1 permits an efficient change of state trigger, since was designed to allow change of state data transfers.

Figure 38 shows the actions which take place during data transfer using a class 1 client transport instance and a class 1 server transport instance.



NOTE The T-PDU packet comprises the transport header from the client transport and data from the application.

Figure 38 – Data flow diagram using client transport class 1 and server transport class 1

Figure 39 shows the sequence in which actions take place when transferring data using client transport class 1 and server transport class 1.

NOTE 2 The sequence count is incremented with every write. Also, in the case where the packet is lost on a new data sample, and the packet is later triggered and sent, the data received by the client is treated as new data. This is because this is the first time the server has received this sequence count. This mechanism provides fault tolerance for those samples that change infrequently.

NOTE 3 It is an implementation choice to decide whether to update the data buffer upon receipt of duplicate data or not, based on the potential impact of the additional processing.

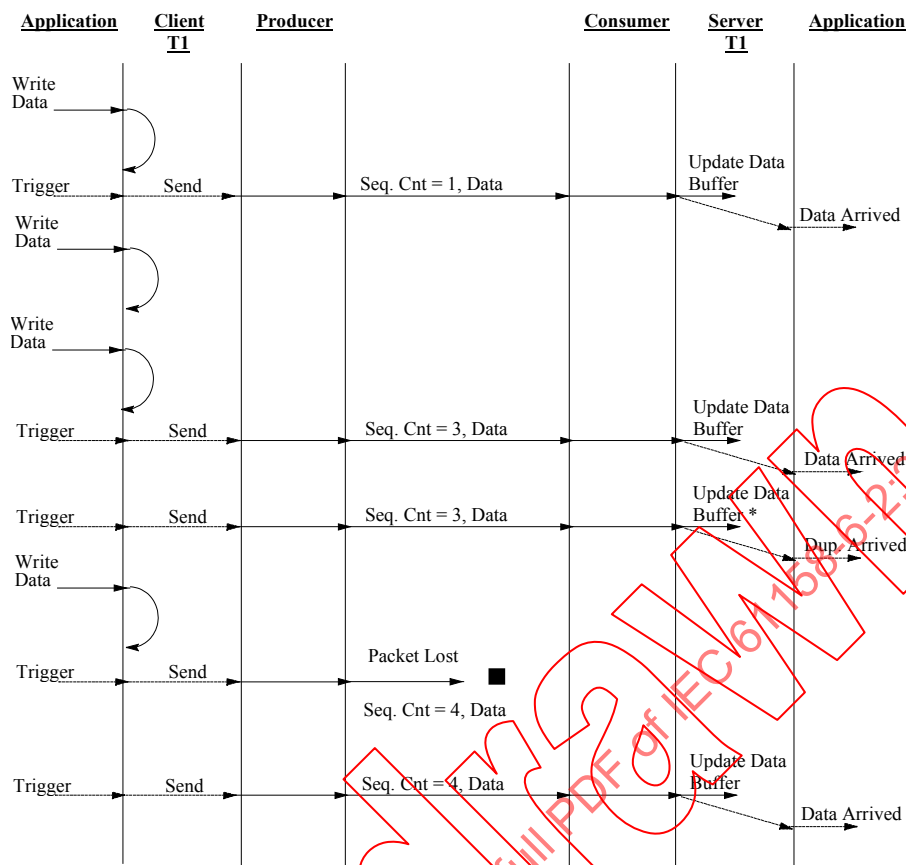


Figure 39 – Sequence diagram of data transfer using client transport class 1 and server transport class 1

Transport class 1 can be used in conjunction with a heartbeat timer to keep a connection open when there are periods when no data is transmitted. The heartbeat timer shall be initialized every time the producing node of the network connection transmits a packet, and shall be set so that its maximum value, or the maximum time between data transmissions, is less than the path time-out value for the network connection. When the heartbeat timer reaches its maximum value, it triggers the client transport instance so that the producing node produces and retransmits the packet in the client-side TPDU buffer before the connection times out. The consuming node of the network connection, recognizing that the packet has the same sequence count as the last packet it received, may or may not overwrite the packet in the server-side TPDU buffer; that implementation decision is left to the product developers, since they can best assess the impact of additional processing.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, a timer may be used to trigger the production of the current data in the TPDU buffer. To transmit new data the application would first write to the TPDU buffer and then trigger the client transport.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, the data arrived and duplicated arrived events may be ORed together to reset a timer. If this timer were to expire, the application would know that no data was received within the designated time. The data arrived event would identify new samples of data. This may allow applications to reduce their overhead by only processing new data samples.

Possible uses of transport class 1 include sampled inputs, standard outputs, cyclic block transfers, diagnostic events, and communication between controllers and operator interface devices.

9.3.6.2 Transport class 1 client

9.3.6.2.1 Functions

Transport class 1 starts with the behaviour in class 0 and adds a sequence count. This sequence count is incremented by the client transport class when data is written to the data buffer. When the transport receives a *write* event, it shall increment the sequence count. When a *trigger* event is received the transport shall update the sequence count in the TPDU buffer and invoke the *send* service.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, a timer may be used to trigger the production of the current data in the TPDU buffer. To transmit new data the application would first write to the TPDU buffer and then trigger the client transport.

A class 1 client transport is responsible for:

- accepting the client application's trigger that it has written a data packet to the client-side TPDU buffer,
- writing a transport header to the TPDU buffer for each packet that the client application writes to the client-side TPDU buffer,
- initializing the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission),
- triggering the producing node of the network connection to produce the TPDU packet on the link and send it to the consuming node of the network connection.

9.3.6.2.2 States

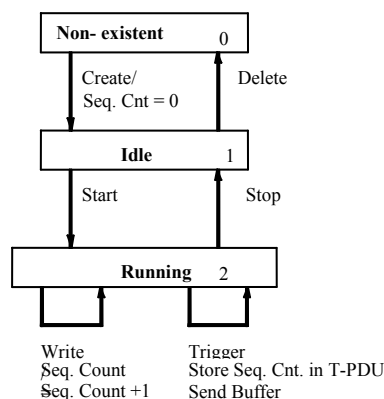
The defined states and their descriptions of the Class 1 Transport Client are listed in Table 231.

Table 231 – Class 1 transport client states

State	Description
Non-existent	The instance of Transport Class 1 does not exist.
Idle	The instance of Transport Class 1 has been created but not yet started.
Running	The instance of Transport Class 1 has been created and started.

9.3.6.2.3 State transitions

State transitions of the class 1 client transport are shown in Figure 40.



NOTE The sequence count is set to 0 by the create service. It is then incremented after every *write* event until it reaches a maximum value of $2^{16}-1$. After that it rolls over to 0.

Figure 40 – Class 1 client STD**9.3.6.2.4 State event matrix**

State transitions of the class 1 client transport are shown in Table 232.

Table 232 – Class 1 client SEM

Event	State		
	Non-existent	Idle	Running
Create	1) Transition to Idle 2) Set seq. count = 0	Error	Error
Delete	Error	Transition to Non-existent	Error
Start	Error	Transition to Running	Error
Stop	Error	No Action	Transition to Idle
Write	Error	No Action	1) Seq. Cnt = Seq. Cnt + 1
Trigger	Error	No Action	1) Store Seq. Cnt. in TPDU Buffer 2) Send

9.3.6.3 Transport class 1 server**9.3.6.3.1 Functions**

Transport class 1 starts with the behaviour in class 0 and adds a sequence count. The received sequence count is compared with the previous sequence count. If they are equal, the received packet is considered to be a duplicate packet. If they are not equal, the received data is considered to be a new sample. There is no attempt to count how many sequence counts have changed between samples.

Applications could use this transport class to distinguish between new samples and old samples that were sent to maintain the connection. To maintain the connection, the data arrived and duplicated arrived events may be ORed together to reset a timer. If this timer were to expire, the application would know that no data was received within the designated time. The data arrived event would identify new samples of data. This may allow applications to reduce their overhead by only processing new data samples.

A class 1 server transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side TPDU buffer,
- locating and reading the transport header of each packet that the consuming node of the network connection writes to the server-side TPDU buffer,
- triggering the server application that data has been written to the server-side TPDU buffer,
- comparing the sequence count of each packet with that of the previous packet,
- notifying the server application that the most recently arrived packet is a duplicate of the previous one when their sequence counts match.

9.3.6.3.2 States

The defined states and their descriptions of the Class 1 Transport Server are listed in Table 233.

Table 233 – Class 1 transport server states

State	Description
Non-existent	The instance of Transport Class 1 does not exist.
Idle	The instance of Transport Class 1 has been created but not yet started.
Ready to run	The instance of Transport Class 1 has been created and started, waiting for a first data packet to arrive.
Running	The instance of Transport Class 1 has been created and started, first data packet has arrived.

9.3.6.3.3 State transitions

State transitions of the class 1 server transport are shown in Figure 41

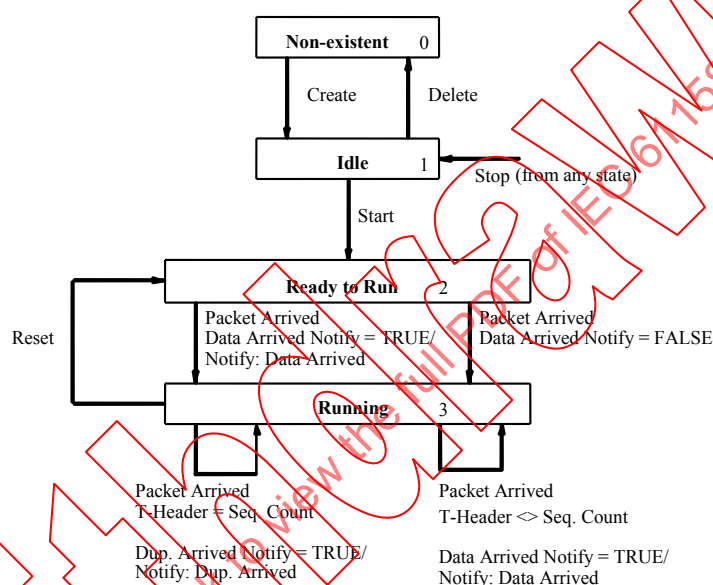


Figure 41 – Class 1 server STD

9.3.6.3.4 State event matrix

State transitions of the class 1 server transport are shown in Table 234.

Table 234 – Class 1 server SEM

Event	State/			
	Non-existent	Idle	Ready to run	Running
Create	Transition to Idle	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error
Start	Error	Transition to Ready to Run	Error	Error
Stop	Error	No action	Transition to Idle	Transition to Idle
Reset	Error	Error	No Action	Transition to Ready to Run
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	Transition to Running Notify: Data Arrived	Notify: Data Arrived
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	Transition to Running Notify: Data Arrived	Notify: Duplicate Arrived
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	Transition to Running Notify: Data Arrived	Notify: Data Arrived
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	Transition to Running Notify: Data Arrived	No action
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	Transition to Running	No action
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	Transition to Running	Notify: Duplicate Arrived
Packet Arrived Transport Header <> Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	Transition to Running	No action
Packet Arrived Transport Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	Transition to Running	No action

9.3.7 Transport state machines – class 2

9.3.7.1 Functions

Transport class 2 starts with the behaviour in class 1 and adds a return connection that acknowledges the delivery of a packet. The producer connection from the client node can be point to point or multipoint. The connection from the server to the client is point to point.

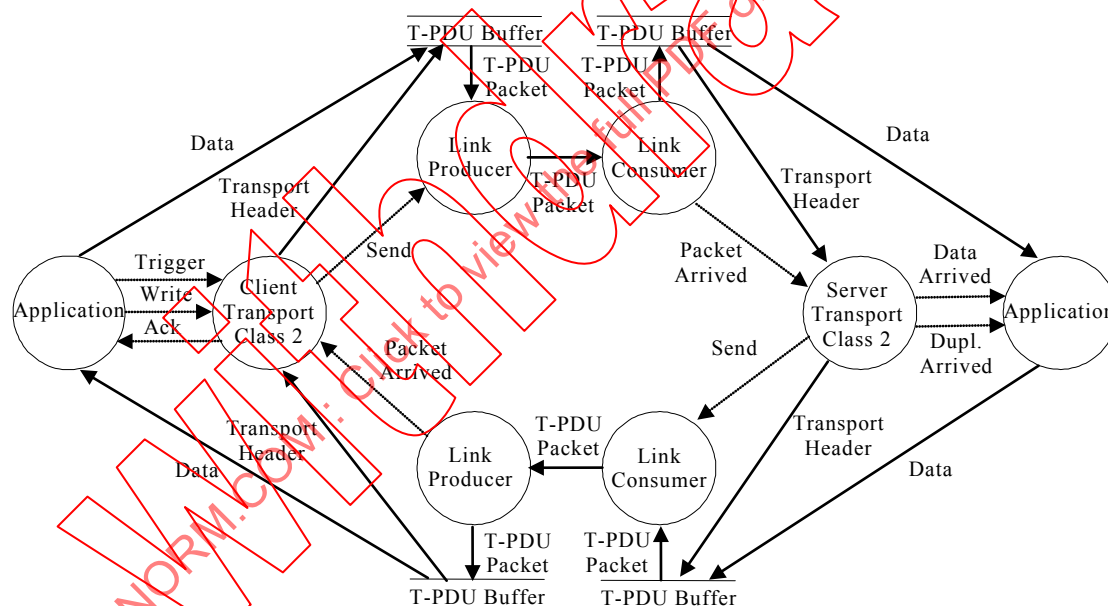
This transport class allows the client application to be notified that the server has received the transmitted packet. This acknowledgement tells the client application only that the data arrived. This does not mean that the server application has read the buffer or that a new sample can be sent without overwriting the buffer.

The main advantage of this class over class 3 (Verified) is that the acknowledgement is returned immediately after the packet arrives at the consumer, while the verify class requires the application to initiate the return verification. The delays associated with receiving the acknowledgement are small, approximately twice the time that is required for a one way transmission.

Transport class 2 uses two network connections: the originator-to-target connection, which can be either point-to-point or multipoint; and the return connection, which shall be point-to-point. A multipoint connection using transport class 2 shall have a point-to-point return connection corresponding to each client-to-server connection. *The acknowledgement notifies the client transport instance that the consuming node of the network connection has received the packet.*

If a server transport instance does not acknowledge receipt of the packet, the client transport instance triggers the client application to retransmit the most recently sent packet; if the client-to-server connection is one of the network connections of a multipoint connection, data is resent over all of the network connections of the multipoint connection. The server application can return data to the client application along with its acknowledgement.

Figure 42 shows the actions which take place during a data transfer using client transport class 2 and server transport class 2



NOTE 1 The client and server transports each use two buffers: one for transmitting, and one for receiving.

NOTE 2 The target application is not involved in sending the acknowledgement.

Figure 42 – Data flow diagram using client transport class 2 and server transport class 2

The server transport instance initiates and sends the acknowledgement to the client transport instance to notify the client application that it can write the next data packet to the client-side TPDU transmit buffer. Each acknowledgement shall be received by the client transport instance before the next packet is sent over the same client-to-server network connection(s).

Figure 43 shows the sequence of actions which take place during data transfer using client transport class 2 and server transport class 2. In the example depicted, the acknowledgement is the only data transmitted in the server-to-client direction. The unshaded areas in Figure 43

indicate client-to-server data transmission, and the shaded areas indicate server-to-client transmission.

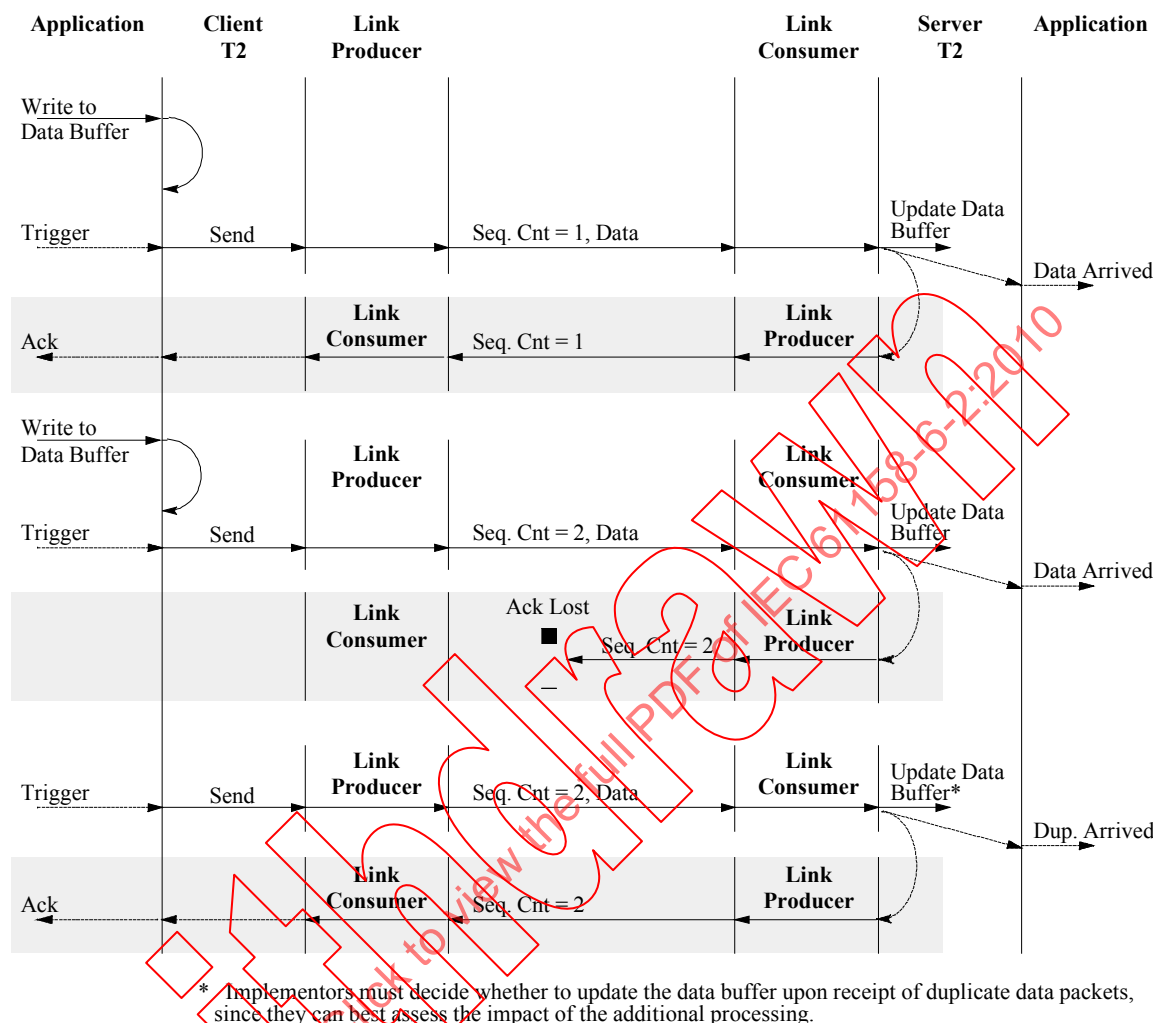


Figure 43 – Diagram of data transfer using client transport class 2 and server transport class 2 without returned data

At times, the server-to-client transmission comprises the acknowledgement *and* additional data.

Figure 44 shows the sequence in which actions take place during data transfer using client transport class 2 and server transport class 2 when the acknowledgement and data are returned to the client.

NOTE The returned data is written asynchronously to the receipt of data from the client.

The unshaded areas in Figure 44 indicate client-to-server data transmission, and the shaded areas indicate server-to-client data transmission.

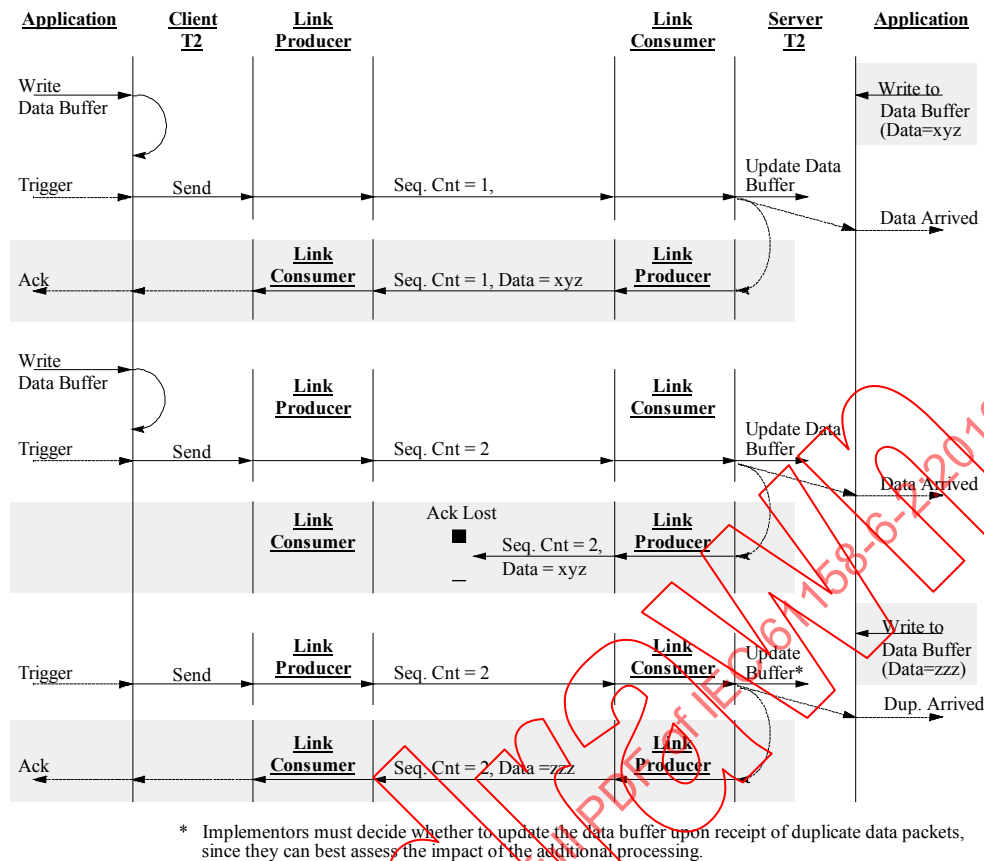


Figure 44 – Sequence diagram of data transfer using client transport class 2 and server transport class 2 with returned data

Possible uses of transport class 2 include master/slave applications and communication between controllers and operator interface devices.

9.3.7.2 Transport class 2 client

9.3.7.2.1 Functions

In the client-to-server direction, a class 2 client transport is responsible for:

- accepting the client application's trigger that it has written a data packet to the client-side TPDU transmit buffer;
- writing a transport header to the client-side TPDU transmit buffer for each packet that the client application writes to it

NOTE The transport header for each packet shall include a sequence count

- initializing the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);
- triggering the producing node of the network connection to produce the TPDU packet on the link and send it to the consuming node of the network connection.

In the server-to-client direction, a class 2 client transport is responsible for:

- accepting notification from the consuming node of the return network connection that it has written data (the server transport instance's acknowledgement) to the client-side TPDU receive buffer;
- locating and reading the transport header of each data packet that the consuming node of the return network connection writes to the client-side TPDU receive buffer;

- notifying the client application that the consuming node of the return network connection has written data to the client-side TPDU receive buffer.

The application can update the data buffer or retrigger data before all of the acknowledgements have been received. If the client transport is retriggered while waiting for an acknowledgement, the previous acknowledgement received register is cleared. This means that the client transport class shall wait for all active consumers to return an acknowledgement before notifying the application.

9.3.7.2.2 States

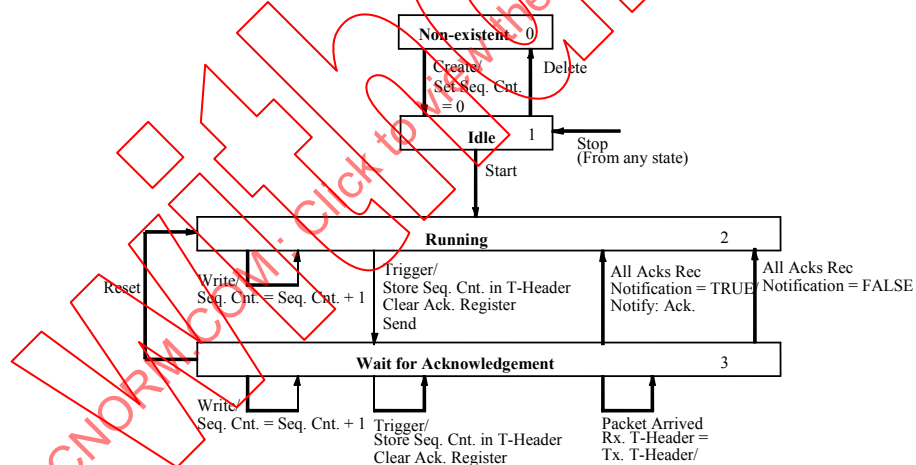
The defined states and their descriptions of the Class 2 Transport Client are listed in Table 235.

Table 235 – Class 2 transport client states

State	Description
Non-existent	The instance of Transport Class 2 does not exist.
Idle	The instance of Transport Class 2 has been created but not yet started.
Running	The instance of Transport Class 2 has been created and started.
Wait for acknowledgement	The instance of Transport Class 2 has been created and started, acknowledgement is pending.

9.3.7.2.3 State transitions

State transitions of the class 2 client transport are shown in Figure 45.



NOTE The sequence count is set to 0 by the create service. It is then incremented after every write event until it reaches a maximum value of $2^{16}-1$. After that it rolls over to 0.

Figure 45 – Class 2 client STD

9.3.7.2.4 State event matrix

Update
Register

Ack.

State transitions of the class 2 client transport are shown in Table 236.

Table 236 – Class 2 client SEM

Event	State			
	Non-existent	Idle	Running	Wait for ack
Create	1) Seq. Cnt. = 0 2) Transition to Idle	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error
Start	Error	Transition to Running	Error	Error
Stop	Error	No Action	Transition to Idle	Transition to Idle
Reset	Error	Error	No Action	Transition to Running
Write	Error	No Action	1) Seq. Cnt. = Seq. Cnt. + 1 2) Write TPDU	1) Seq. Cnt. = Seq. Cnt. + 1 2) Write TPDU
Trigger	Error	No Action	1) Store Seq. Cnt. in TPDU 2) Clear Ack. register 3) Send 4) Transition to Wait for Ack.	1) Store Seq. Cnt. in TPDU 2) Clear Ack. Register 3) Send
Packet Arrived Rx. T-Header = Tx. T-Header	Error	No Action	No Action	1) Update Ack. Register
All Ack Received Ack. Notify = TRUE	Error	No Action	Error	1) Notify: Ack. 2) Transition to Running
All Ack Received Ack. Notify = FALSE	Error	No Action	Error	1) Transition to Running

NOTE The STD and SEM for client classes 2 and 3 are nearly identical. Verify has been substituted for acknowledgement.

9.3.7.3 Transport class 2 server

9.3.7.3.1 Functions

Server transport class 2 starts with the behaviour from transport class 1 and adds a return connection from the server to the client that is used to acknowledge a message. This return connection is used to identify the acknowledgement message.

After receiving a *packet arrived* event, the server transport shall read the received transport header, store it in the transmit transport header and invoke the *send* service. The application is not involved in the acknowledgement. The producer connection from the client node can be point to point or multipoint. The connection from the server to the client is point to point.

In the client-to-server direction, a class 2 server transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side TPDU receive buffer;
- locating and reading the transport header of each packet that the consuming node of the network connection writes to the server-side TPDU receive buffer;
- comparing the sequence count of each packet in the server-side TPDU receive buffer with that of the previous packet;
- notifying the server application that the most recently arrived packet is a duplicate of the previous one when their sequence counts match;
- notifying the server application that data has been written to the server-side TPDU receive buffer. (ACK may contain server data).

In the server-to-client direction, a class 2 server transport is responsible for:

- writing the transport header (including sequence count) of the data in the server-side TPDU receive buffer to the server-side TPDU transmit buffer;
- triggering the producing node of the return network connection to produce the TPDU packet on the link and send it to the consuming node of the return network connection.

9.3.7.3.2 States

The defined states and their descriptions of the Class 2 Transport Server are listed in Table 237.

Table 237 – Class 2 transport server states

State	Description
Non-existent	The instance of Transport Class 2 does not exist.
Idle	The instance of Transport Class 2 has been created but not yet started.
Running	The instance of Transport Class 2 has been created and started.

9.3.7.3.3 State transitions

State transitions of the class 2 server transport are shown in Figure 46.

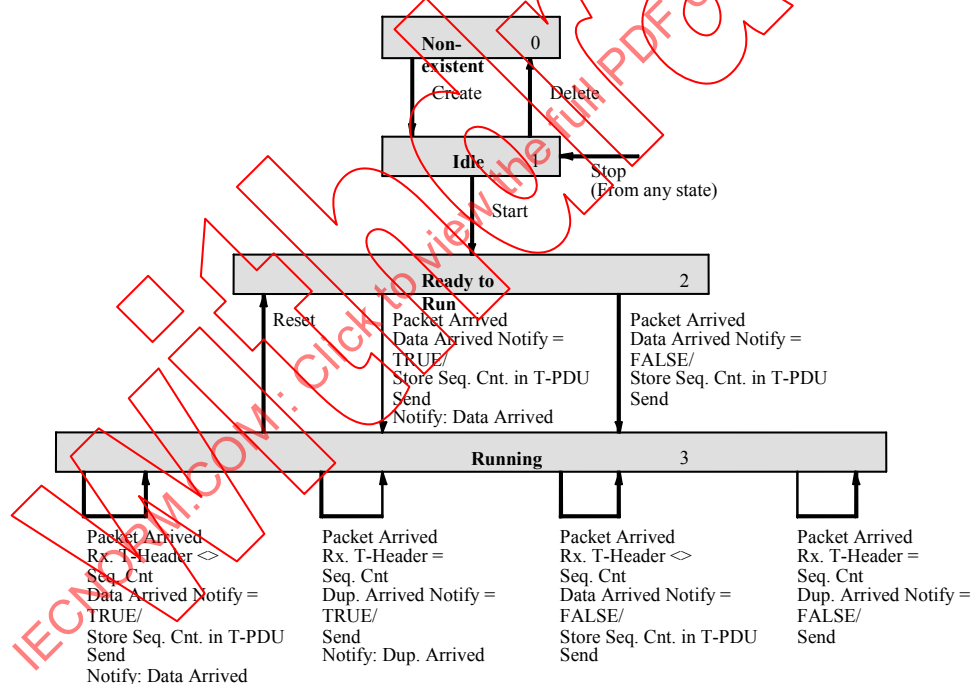


Figure 46 – Class 2 server STD

NOTE The send service is invoked for sequence counts that match and sequence counts that differ. This behaviour is required to ensure that a node that has sent an acknowledgement that was lost shall retransmit the acknowledgement on a retry.

9.3.7.3.4 State event matrix

State transitions of the class 2 server transport are shown in Table 238.

Table 238 – Class 2 server SEM

Event	State			
	Non-existent	Idle	Ready to run	Running
Create	Transition to Idle	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error
Start	Error	Transition to Ready to Run	Error	Error
Stop	Error	No action	Transition to Idle	Transition to Idle
Reset	Error	Error	No action	Transition to Ready to Run
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Notify: Data Arrived 4) Transition to Running	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Notify: Data Arrived
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Notify: Data Arrived 4) Transition to Running	1) Send (Implies Ack) 2) Notify: Duplicate Arrived
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Notify: Data Arrived 4) Transition to Running	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Notify: Data Arrived
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = TRUE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Notify: Data Arrived 4) Transition to Running	1) Send (Implies Ack)
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Transition to Running	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack)
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Transition to Running	1) Send (Implies Ack) 2) Notify: Duplicate Arrived
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Transition to Running	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack)
Packet Arrived Rx. T-Header = Seq. Count Data Arrived Notify = FALSE Duplicate Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in Tx. TPDU 2) Send (Implies Ack) 3) Transition to Running	1) Send (Implies Ack)

9.3.8 Transport state machines – class 3

9.3.8.1 Functions

Transport class 3 starts with the behaviour in class 1 and adds a return connection that verifies that the application has received the packet. The sequence count that shall be returned with the verify is the same as the sequence count sent with the data. This return connection is used to identify the verification message.

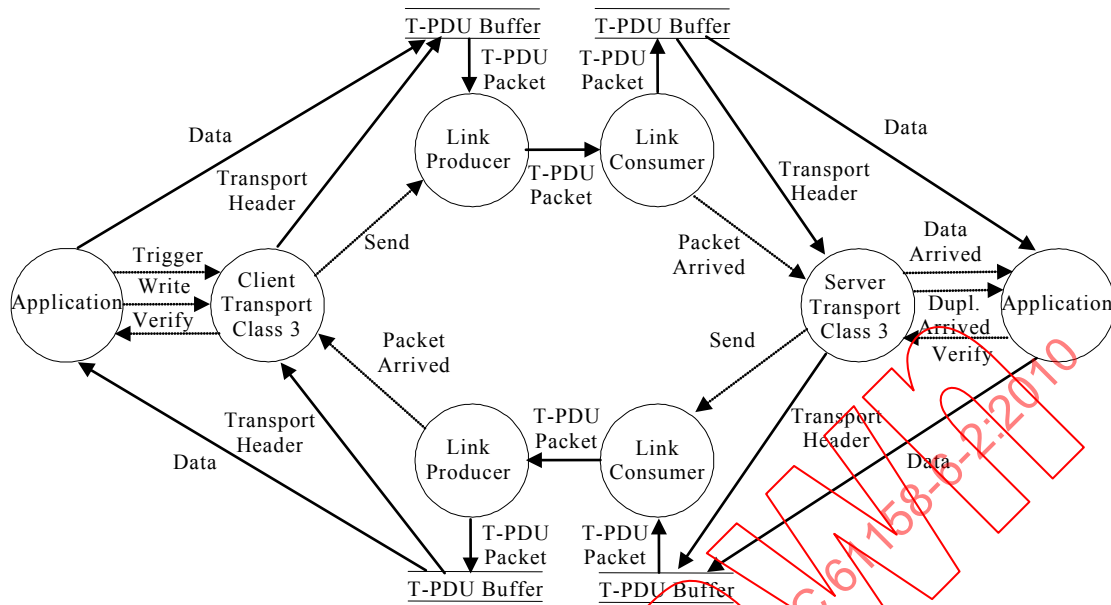
This transport class allows the application to be notified that the server application has responded to the transmitted packet. This verification tells the client application not only that the data arrived but that the server application has read the buffer and that a new sample can be sent without overwriting the buffer.

The main advantage of this class over class 2 (Acknowledgement) is that verification conveys an application response. The disadvantage is that the delay in receiving the verification can be very difficult to predict.

Transport class 3 uses two connections: the originator-to-target connection, which can be either point-to-point or multipoint; and the return connection, which shall be point-to-point. A multipoint connection using transport class 3 shall have a point-to-point return connection corresponding to each client-to-server connection. The verification notifies the client application that the server application has received and read the transmitted data. If a server application does not verify receipt of the packet, the client application retransmits the most recently sent packet; if the client-to-server connection is a multipoint connection, the data is retransmitted over all of the network connections of the multipoint connection.

The server application initiates and sends the verification *after* the consuming node of the network connection has written the packet to the TPDU buffer, *after* the transport has read the packet's transport header, and *after* the application has read the packet. The server's verification signals the client application that the server-side TPDU receive buffer is ready to accept the next data packet.

Figure 47 shows the actions which take place during a data transfer using client transport class 3 and server transport class 3.



NOTE 1 The client and server transports each use two buffers, one for transmitting, and one for receiving.

NOTE 2 The server application can return data to the client application using the connection established for the verification.

Figure 47 – Data flow diagram using client transport class 3 and server transport class 3

Figure 48 shows the sequence of actions which take place during data transfer using client transport class 3 and server transport class 3. In the example depicted, the verification is the only data transmitted in the server-to-client direction. The unshaded areas in Figure 48 indicate client-to-server data transmission, and the shaded areas indicate server-to-client transmission.

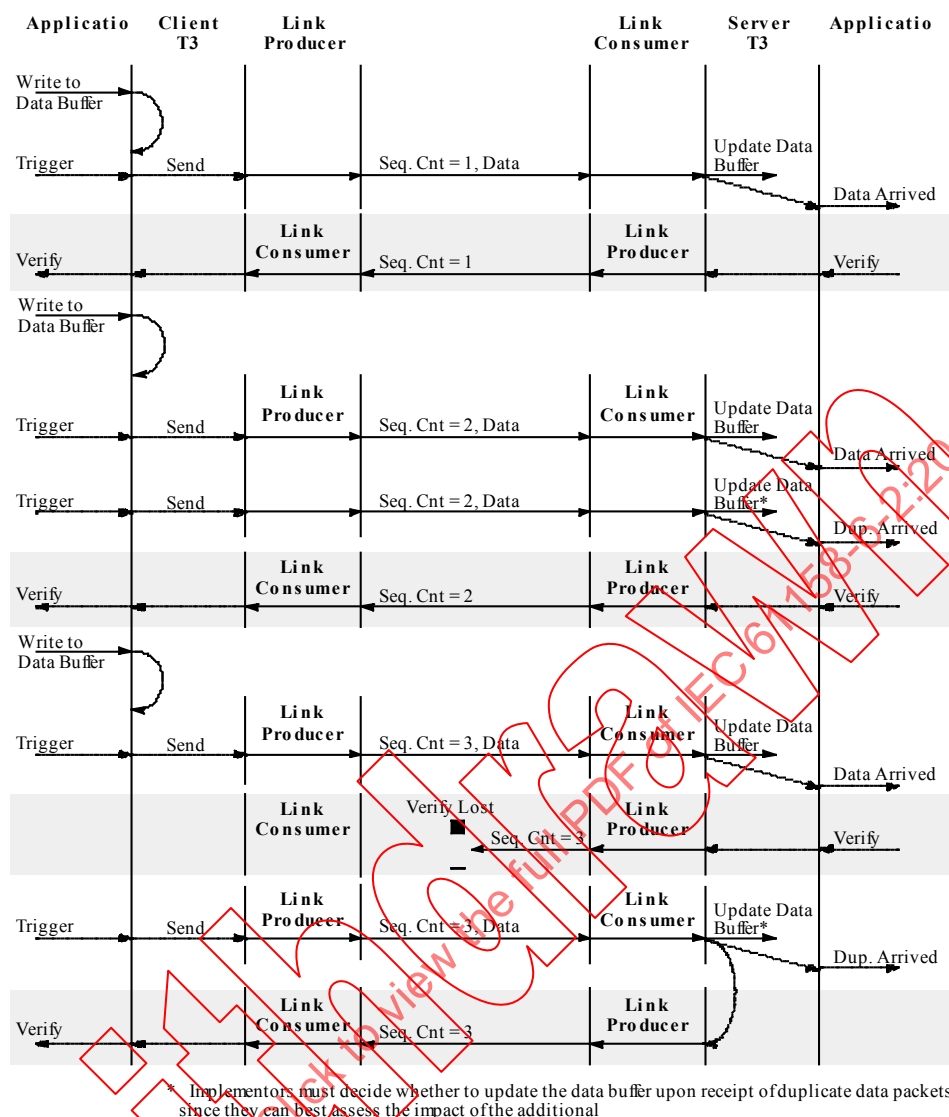


Figure 48 – Sequence diagram of data transfer using client transport class 3 and server transport class 3 *without* returned data

At times, the server-to-client transmission comprises the verification *and* additional data. Figure 49 shows the sequence in which actions take place during data transfer using client transport class 3 and server transport class 3 when the verification and data are returned to the client.

NOTE The returned data is written asynchronously to the receipt of data from the client.

The unshaded areas in Figure 49 indicate client-to-server data transmission, and the shaded areas indicate server-to-client transmission.

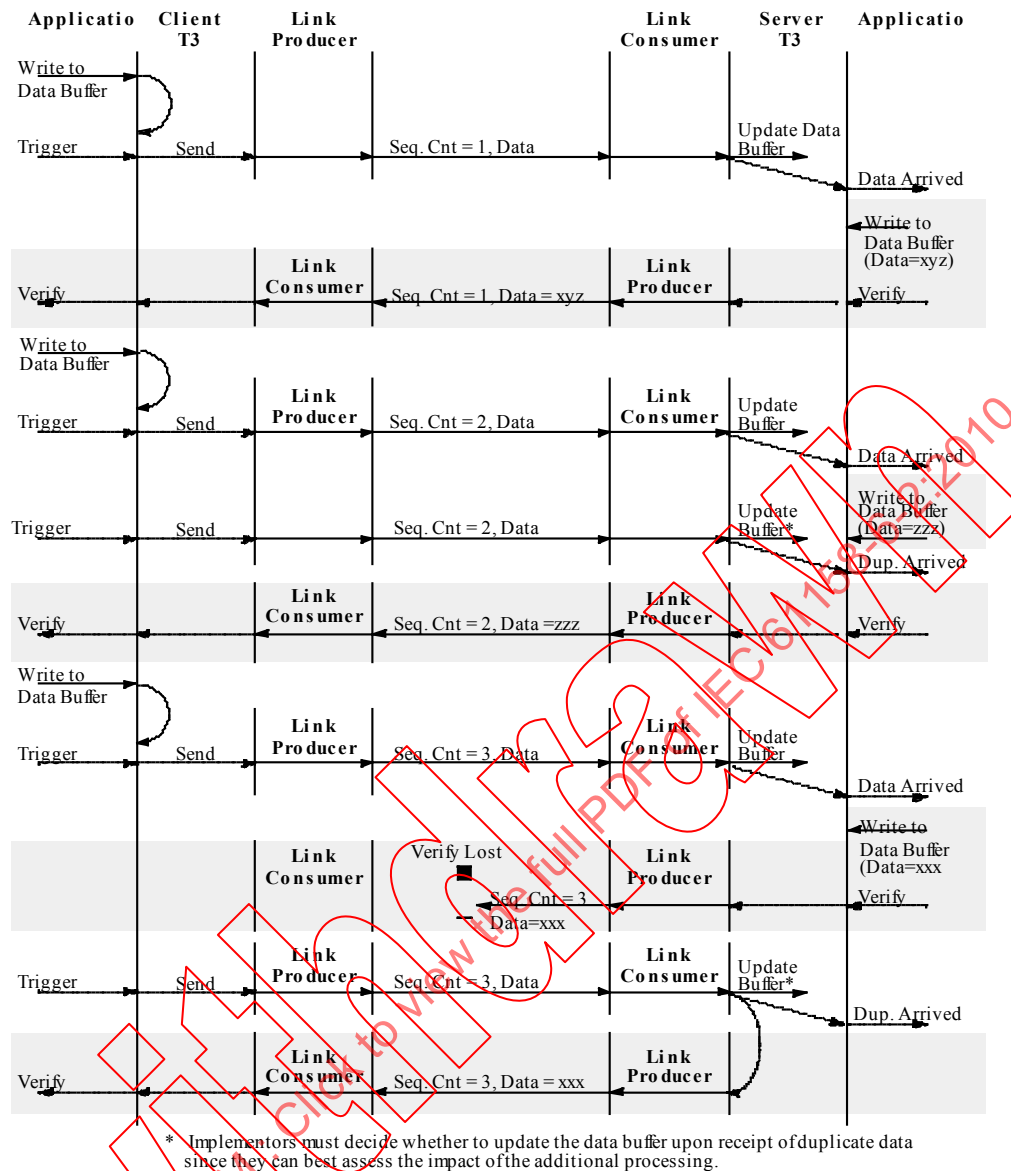


Figure 49 – Sequence diagram of data transfer using client transport class 3 and server transport class 3 with returned data

Possible uses of transport class 3 include change-of-state inputs and outputs, asynchronous [independent] block transfers, event acknowledgements, communication between controllers and operator interface devices, uploads, downloads, and messaging.

9.3.8.2 Transport class 3 client

9.3.8.2.1 Functions

In the client-to-server direction, a class 3 client transport is responsible for:

- accepting the client application's trigger that it has written a data packet to the client-side TPDU transmit buffer;
- writing a transport header to the client-side TPDU transmit buffer for each packet that the client application writes to it;
- initializing the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);

- triggering the producing node of the network connection to produce the TPDU packet on the link and send it to the consuming node of the network connection.

In the server-to-client direction, a class 3 client transport is responsible for:

- accepting notification from the consuming node of the return network connection that it has written data (the server transport instance's verification) to the client-side TPDU receive buffer;
- locating and reading the transport header of each data packet that the consuming node of the return network connection writes to the client-side TPDU receive buffer;
- notifying the client application that the consuming node of the return network connection has written data to the client-side TPDU receive buffer.

The application can update the data buffer or retrigger data before all of the verifications have been received. If the client transport is retriggered while waiting for verification, the previous verification received register is cleared. This means that the client transport class shall wait for all active consumers to return verification before notifying the application.

9.3.8.2.2 States

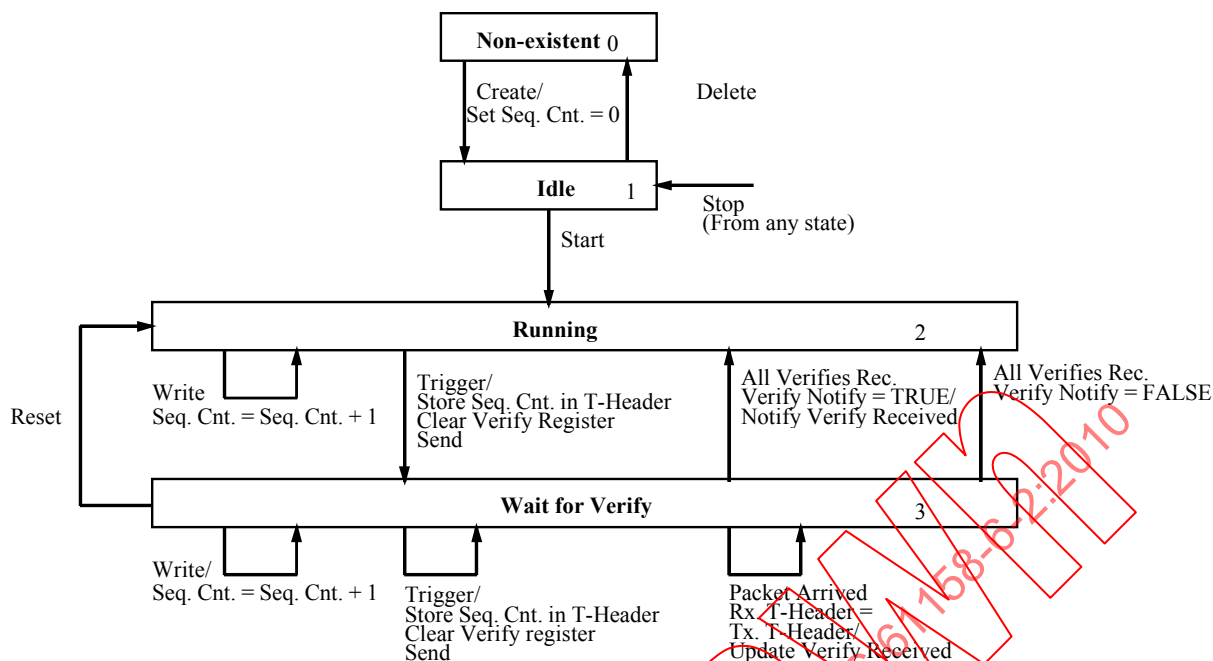
The defined states and their descriptions of the Class 3 Transport Client are listed in Table 239.

Table 239 – Class 3 transport client states

State	Description
Non-existent	The instance of Transport Class 3 does not exist.
Idle	The instance of Transport Class 3 has been created but not yet started.
Running	The instance of Transport Class 3 has been created and started.
Wait to verify	The instance of Transport Class 3 has been created and started, verification is pending.

9.3.8.2.3 State transitions

State transitions of the class 3 client transport are shown in Figure 50.



NOTE The sequence count is set to 0 by the create service. It is then incremented after every *write* event until it reaches a maximum value of $2^{16}-1$. After that it rolls over to 0.

Figure 50 – Class 3 client STD

9.3.8.2.4 State event matrix

State transitions of the class 3 client transport are shown in Table 240.

Table 240 – Class 3 client SEM

Event	State			
	Non-existent	Idle	Running	Wait for verify
Create	1) Seq. Cnt. = 0 2) Transition to Idle	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error
Start	Error	Transition to Running	Error	Error
Stop	Error	No action	Transition to Idle	Transition to Idle
Reset	Error	Error	No Action	Transition to Running
Write	Error	No action	Seq. Cnt. = Seq. Cnt. + 1	Seq. Cnt. = Seq. Cnt. + 1
Trigger	Error	No action	1) Store Seq. Cnt. in TPDU 2) Clear Verify register 3) Send 4) Transition to Wait for Verify	1) Store Seq. Cnt. in TPDU 2) Clear Verify register 3) Send
Packet Arrived Rx. T- header = Tx. T-Header	No action	No action	No Action	1) Update Verify Register
All Verifies Received Notification = TRUE	No action	No action	Error	1) Notify: Verify 2) Transition to Running
All Verifies Received Notification = FALSE	No action	No action	Error	1) Transition to Running

NOTE The STD and SEM for client classes 2 and 3 are nearly identical. Acknowledgement has been substituted for verification.

9.3.8.3 Transport class 3 server

9.3.8.3.1 Functions

Transport class 3 starts with the behaviour in class 1 and adds a return connection that verifies that the application has received the packet. It is the responsibility of the application to verify receipt of data. The sequence count that shall be returned with the verification is the same as the sequence count sent with the data.

The application is required to read the TPDU buffer before invoking the verify service.

In the client-to-server direction, a class 3 server transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side TPDU receive buffer;
- locating and reading the transport header of each packet that the consuming node of the network connection writes to the server-side TPDU receive buffer;
- comparing the sequence count of each packet in the server-side TPDU receive buffer with that of the previous packet;
- notifying the server application that the most recently arrived packet is a duplicate of the previous one when their sequence counts match;
- notifying the server application that data has been written to the server-side TPDU receive buffer.

In the server-to-client direction, a class 3 server transport is responsible for:

- writing the transport header of the data in the server-side TPDU receive buffer to the server-side TPDU transmit buffer;
- correlating that transport header with any data that the server application wants to send to the client application and has written to the server-side TPDU transmit buffer;
- triggering the producing node of the return network connection to produce the TPDU packet on the link and send it to the consuming node of the return network connection.

NOTE Data from the application can be return from the server to the client along the verification connection.

The send service is invoked when verify is received from an application or when a duplicate transmission is received while in the running state. Retransmitting when a previously verified packet has been received allows this class to recover from lost verifications.

9.3.8.3.2 States

The defined states and their descriptions of the Class 3 Transport Server are listed in Table 241.

Table 241 – Class 3 transport server states

State	Description
Non-existent	The instance of Transport Class 3 does not exist.
Idle	The instance of Transport Class 3 has been created but not yet started.
Read to run	The instance of Transport Class 3 has been created and started, waiting for the first packet to arrive.
Wait for verify	The instance of Transport Class 3 has been created and started, the first packet arrived, waiting for verification to be requested by the application.
Running	The instance of Transport Class 3 has been created and started, verification has been processed.

9.3.8.3.3 State transitions

State transitions of the class 3 server transport are shown in Figure 51.

Withdrawing
IECNORM.COM: Click to view the full PDF of IEC 61158-6-2:2010

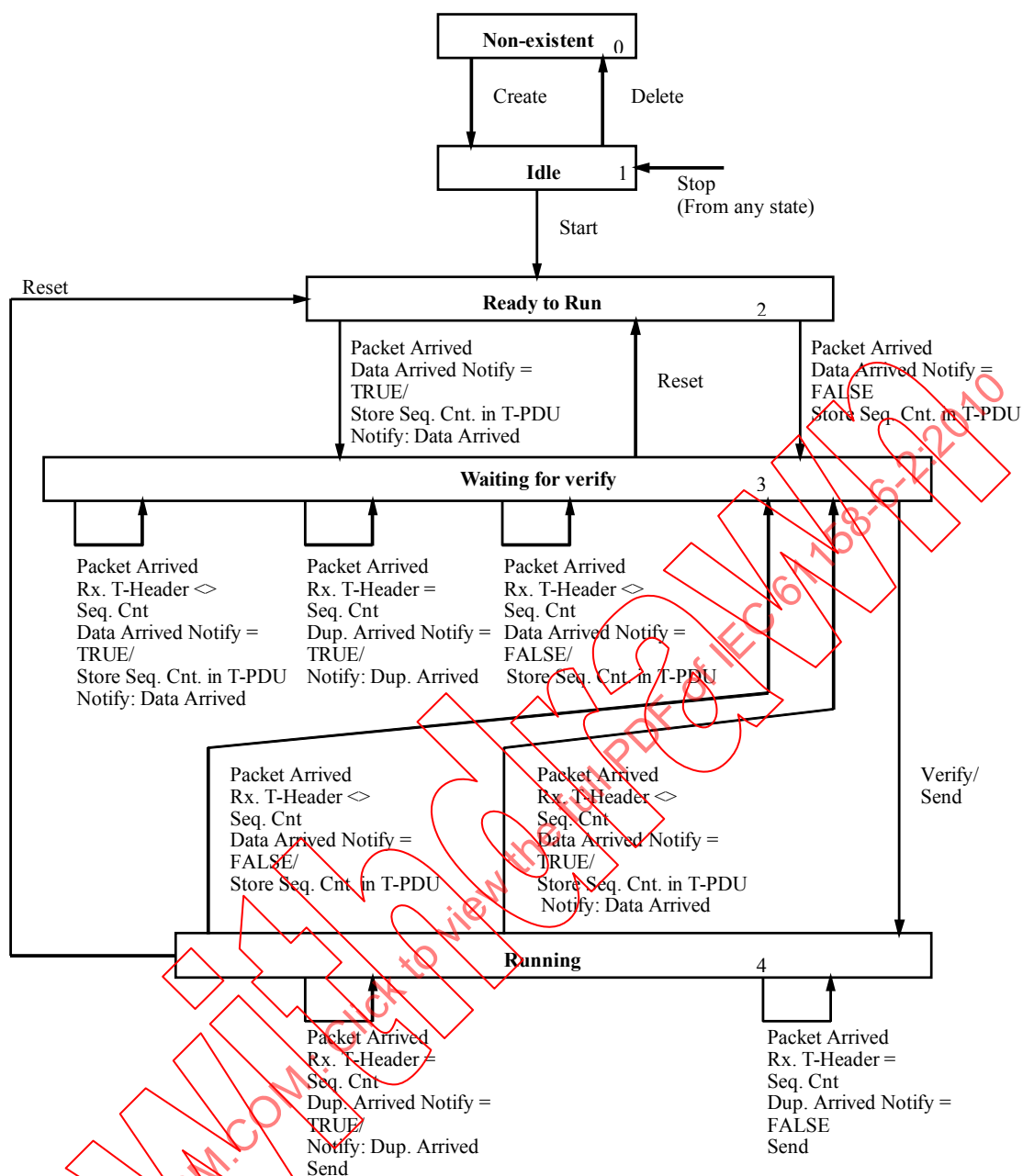


Figure 51 – Class 3 server STD

9.3.8.3.4 State event matrix

State transitions of the class 3 server transport are shown in Table 242.

Table 242 – Class 3 server SEM

Event	State				
	Non-existent	Idle	Ready to run	Waiting for verify	Running
Create	Transition to Idle	Error	Error	Error	Error
Delete	Error	Transition to Non-existent	Error	Error	Error
Start	Error	Transition to Ready to Run	Error	Error	Error
Stop	Error	No action	Transition to Idle	Transition to Idle	Transition to Idle
Reset	Error	Error	No action	Transition to Ready to Run	Transition to Ready to Run
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = TRUE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq.Count Data Arrived Notify = TRUE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	1) Notify: Dup. Arrived	1) Notify: Dup. Arrived 2) Send (Implies Verify)
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = TRUE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq.Count Data Arrived Notify = TRUE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Notify: Data Arrived 3) Transition to Waiting for Verify	No action	1) Send (Implies Verify)
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = FALSE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify	1) Store Seq. Cnt. in TPDU	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq.Count Data Arrived Notify = FALSE Dup Arrived Notify = TRUE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify	1) Notify: Dup. Arrived	1) Notify: Dup. Arrived 2) Send (Implies Verify)
Packet Arrived Rx. T-Header <> Seq.Count Data Arrived Notify = FALSE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify	1) Store Seq. Cnt. in TPDU	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify
Packet Arrived Rx. T-Header = Seq.Count Data Arrived Notify = FALSE Dup Arrived Notify = FALSE	No action	No action	1) Store Seq. Cnt. in TPDU 2) Transition to Waiting for Verify	No action	1) Send (Implies Verify)
Verify	Error	No action	No action	1) Send (Implies Verify) 2) Transition to Running	No action

9.3.9 Transport state machines – classes 4, 5, 6

9.3.9.1 General

This subclause explains the common elements of the transport classes 4 – 6, which provide higher-level features such as non-blocking, fragmentation/assembly, and transport-level acknowledgement. Transport classes 4 and 5 shall not be used with messaging. Messaging does not provide a way to match requests with responses. Use class 3 for messaging without fragmentation.

The fragmentation protocols assume that per-connection send order is preserved throughout the system; overwrite is permissible, but changing the order of the messages is not.

9.3.9.2 Class 4 and 5 data flow diagram

Figure 52 shows the relationship of the class 4 and 5 Transport to the Application and the Link Producer and Link Consumer, the data that move between these functions, and the events which are generated and used by them. (The class 6 data flow diagram is included with the class 6 definition, see 9.3.12.2.) There are some significant differences between these and the class 3 transport.

- The application data is entered into a queue, not placed directly into the TPDU Buffer. This allows the application to give several requests to the transport without waiting for the acknowledgement, verification, or response to a previous request.
- The success or failure of an application request is placed into a queue for the application to access, rather than using a simple event. This is done because the execution of the application and the delivery of the application data are asynchronous.
- The events and data on the right hand side and the left hand side are identical. There is not a “Client” and “Server” transport for these classes. Both sides are capable of sending and receiving application data.
- The trigger timer is explicitly identified, rather than being a portion of the application. This is because the transport is using the trigger signal to perform timing functions.

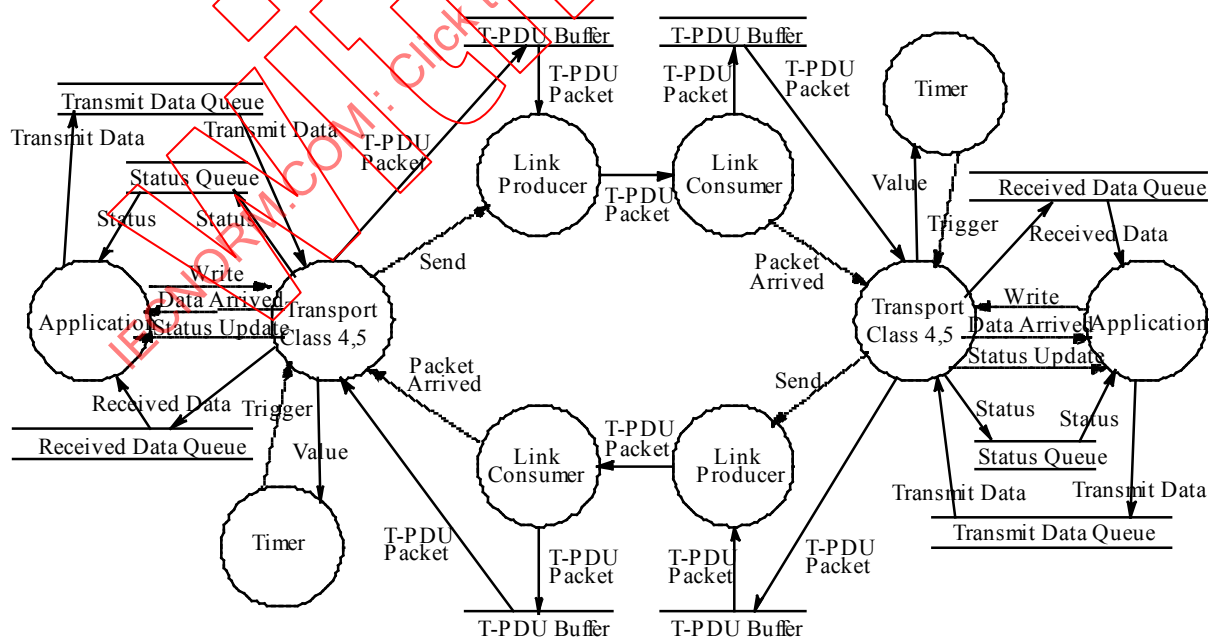


Figure 52 – Data flow diagram using transport classes 4 and 5

The Write event is used to notify the transport that new application data has been placed in the queue. The Trigger event is assumed to be provided by a resettable retry timer. These

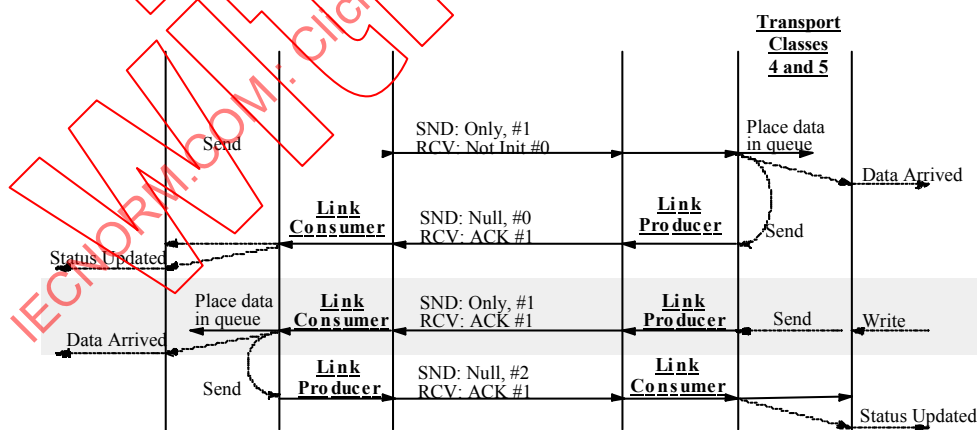
Table 243 – Write and trigger events in class 4 and 5 transport

	Classes 0 to 3	Classes 4 to 6
Write event	"New data available": Causes sequence count to increment.	"New data available": Indicates queue is non-empty. If the transport is not otherwise busy, shall cause data to be sent.
Trigger event	Causes data to be sent.	Causes retry or keep alive to be sent.

9.3.9.3.1 Typical sequences

9.3.9.3.2 Typical message exchange sequence

When a transport has some data to send, it uses the *Only* command and when it has nothing to send, it uses the *Null* command. It is recommended that the Sender state machine send zero application data octets with the *Null* command. It is also recommended that the Receiver state machine accept *Null* commands with a non-zero number of application data octets, which it ignores.



9.3.9.3.3 Typical sequence with overwrite

It is permissible for a transport to send two data packets in a row. This implies that the second one might overwrite the first one in an intermediate module. This is desirable, since the second packet contains information that is newer than the first one and the data in the first one is no longer needed.

In the previous example (see 9.3.9.3.2), the transport on the right had nothing to send when it received the *Only* #1 from the transport on the left. It needed to send the *Null* #0, so that it could deliver the ACK #1. But when the application on the right did a WRITE, the transport immediately sent the *Only* #1 along with the ACK #1. The previous example shows both of them arriving separately. Figure 54 shows the left side's *Only* #1 overwriting the *Null* #0.

Figure 54 also shows a different situation where an overwrite can occur. The right side sends a *Null* #2 with the ACK #2 as the result of the keep-alive *Trigger* event. But, before it can be delivered, the right side receives the *Only* #3 from the left side. The left side immediately sends the *Null* #2, but this time with the newer ACK #3, which overwrites the ACK #2.

It is not necessary to wait for the ACK to a *Null* before sending an *Only*.

When sending an *Only*, it shall have a different sequence number than the *Null*, so sender knows which is being ACK'd by the other side.

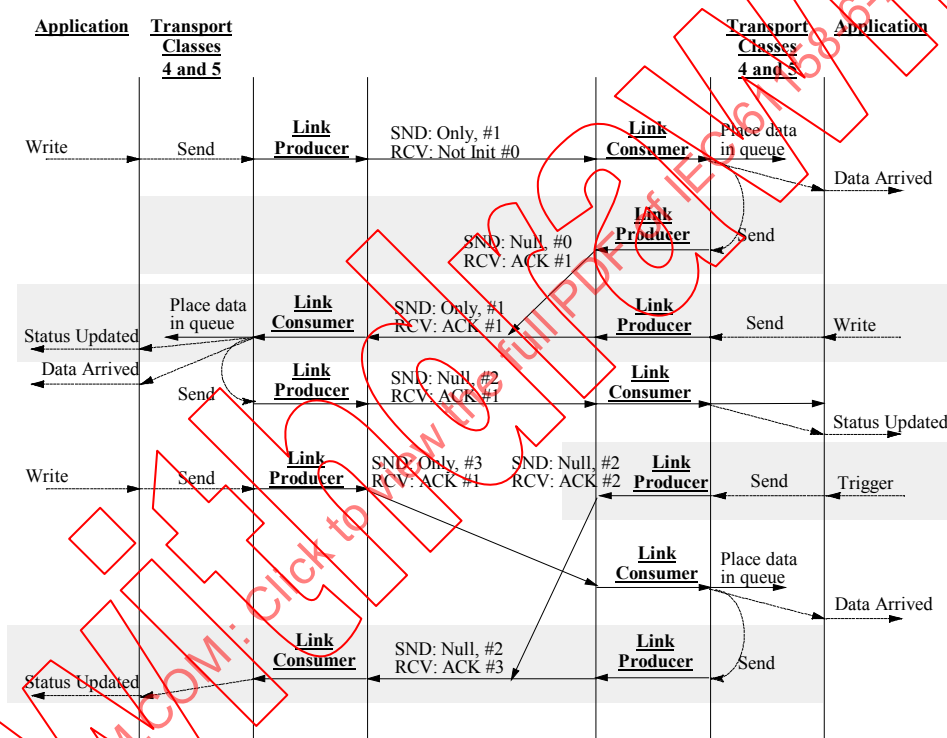


Figure 54 – Sequence diagram of messages overwriting each other

9.3.9.3.4 Typical queued sequence

Figure 55 shows the sequence in which actions take place during a messaging request/response exchange using transport classes 4 and 5. In this case, the applications place some data into the send queue before the first data transfer has been acknowledged. The unshaded areas in Figure 55 indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

In this case, the left transport shall not use the *Null* #2 to deliver the ACK #1, but shall send the data from its queue via the *Only* #2.

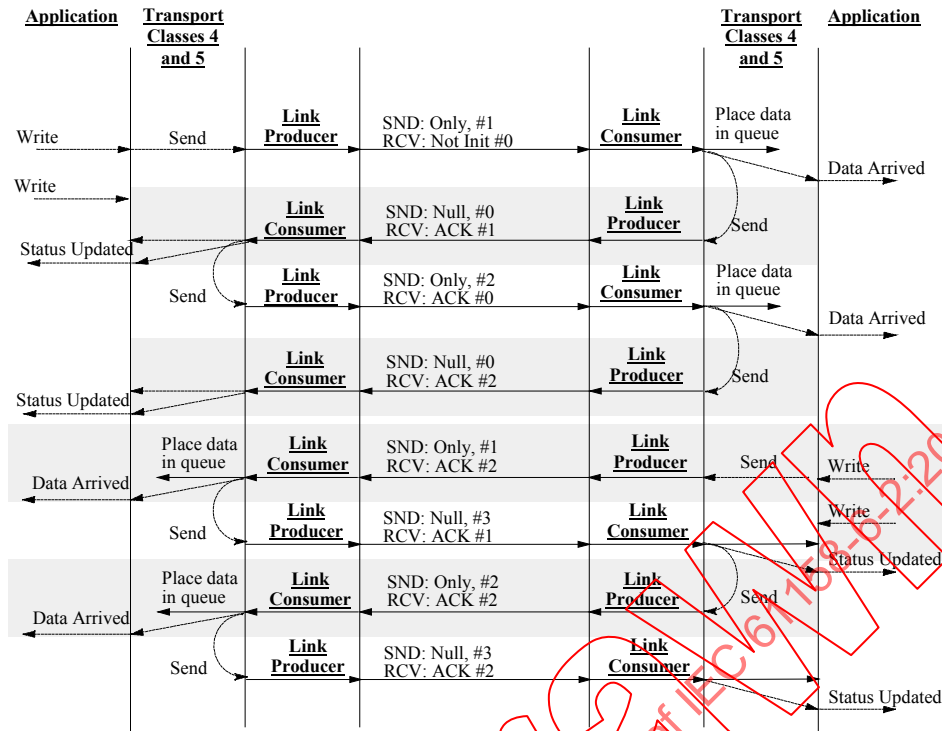


Figure 55 – Sequence diagram of queued message exchange using transport classes 4 and 5

9.3.9.3.5 Typical sequence with retries

Retries are performed when a transport does not receive a Receiver header with the same sequence count as its current Sender header in the timeout period. The TPDU is not changed for a retry. Figure 56 shows two retry situations: where an *Only* command fails to be delivered and where a *Null* with an expected ACK fails to be delivered. The Trigger event is generated by the retry timer.

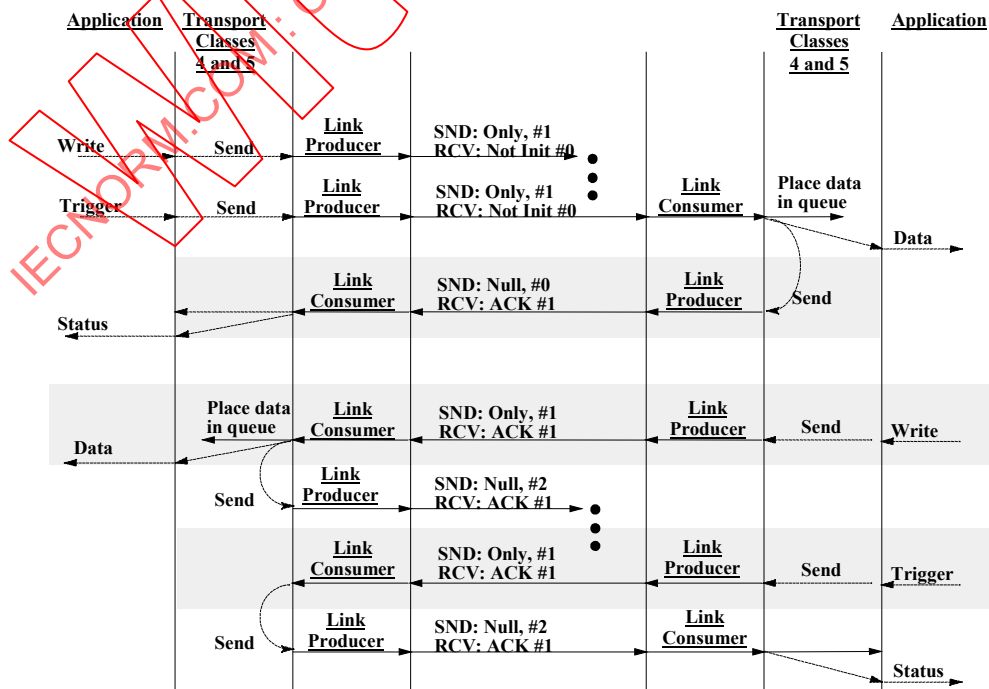


Figure 56 – Sequence diagram of retries using transport classes 4 and 5

9.3.9.3.6 Typical idle sequence

When a class 4 or 5 transport has nothing to send which shall impact the other state machine, it does not need to send anything. However, it is necessary to send some small amount of traffic to keep the connection alive. The Trigger event shall be used, with the timer set to a “keep alive” value. Figure 57 shows the *Null* command used to keep the connection alive, using the Trigger event.

NOTE the *Null* command does not need to be acknowledged. The other transport shall not retry a *Null* command. It shall continue to resend the *Null* command at the “keep alive” rate.

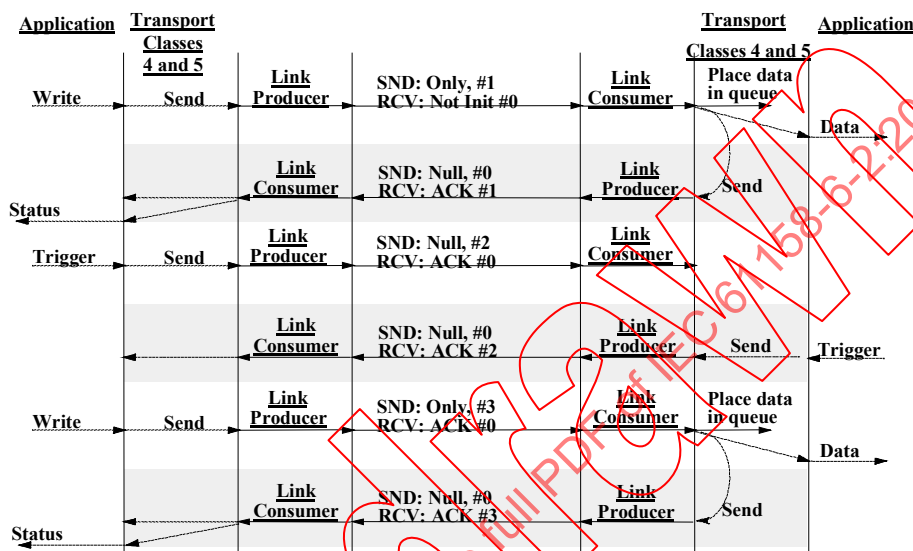


Figure 57 – Sequence diagram of idle traffic using transport classes 4 and 5

9.3.9.4 Classes 4 to 6 basic structure

To make the specification of the Non-Blocking classes (Classes 4 and 5) easier to read, the specification describes them as being made up of two state machines: one to handle the “Sender” function (Transport PDU’s from this device to the other) and the other to handle the “Receiver” function (Transport PDU’s from the other device to this one). (The class 6 transport has a client and a server state machine similar to the class 3 transport.)

The Sender and Receiver state machines in one node are coupled to each other via one event and three data items. The event and the data items come from the Receiver state machine to the Sender one. The *NewData* event indicates to the Sender state machine that one or more of the data items have changed. The *To_Send* and *From_Recv* data items are copies of the headers as shown in the diagram. The *Flag* data item is used by the Receiver state machine to tell the Sender state machine that it shall generate a *Send* event to the link producer, even if the Sender one has no reason to do so. This usually is accompanied by a change in the *From_Recv* data item, but not all changes in *From_Recv* need to be sent out immediately.

The Sender state machine has control of the Link Producer; the Receiver state machine accepts the data from the Link Consumer. The Sender state machine accepts data from the application; the Receiver state machine delivers data to the application. (This application data might be either Application Requests or Responses or data; the transport does not care nor know which it is.)

Figure 58 shows the relationship of these state machines. Both state machines are contained in one end node. The other end is identical.

Figure 59 shows the basic structure for class 6.

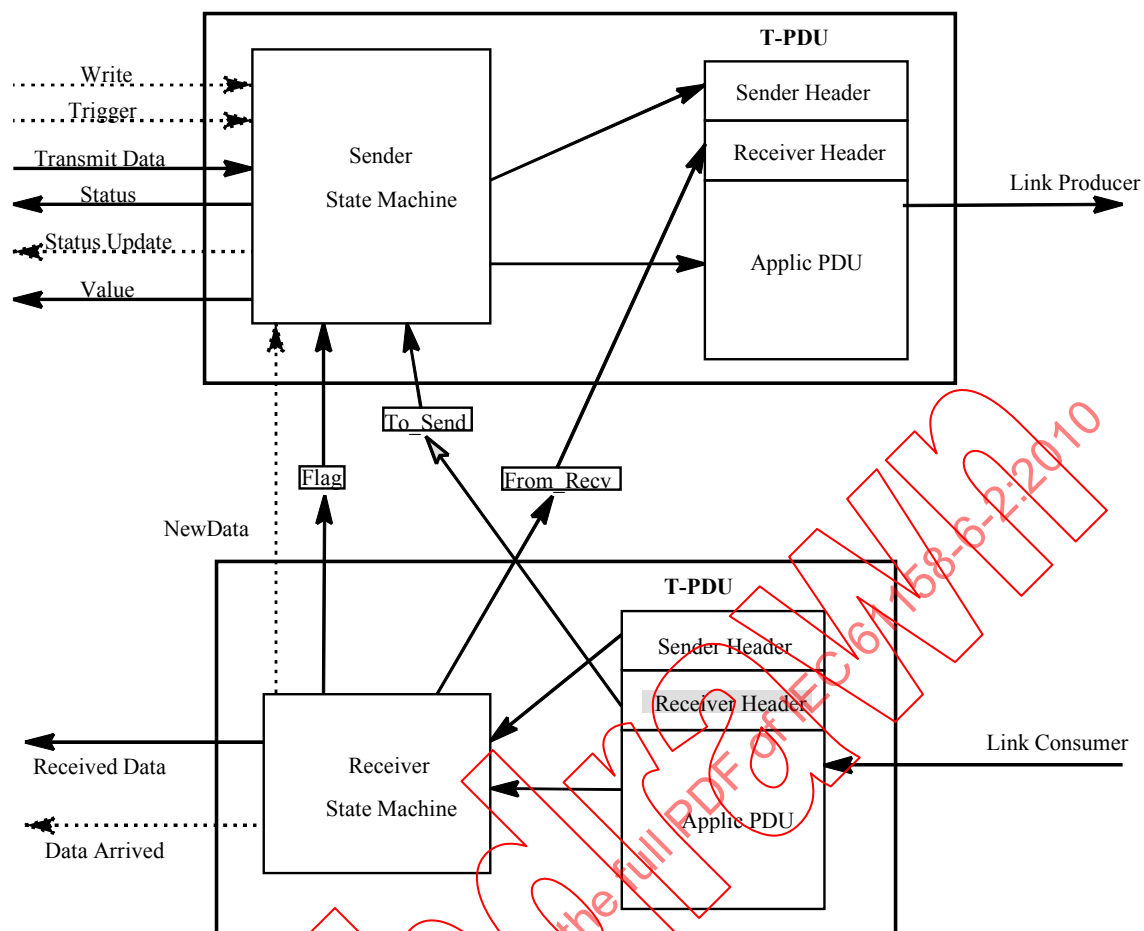


Figure 58 – Classes 4 and 5 basic structure

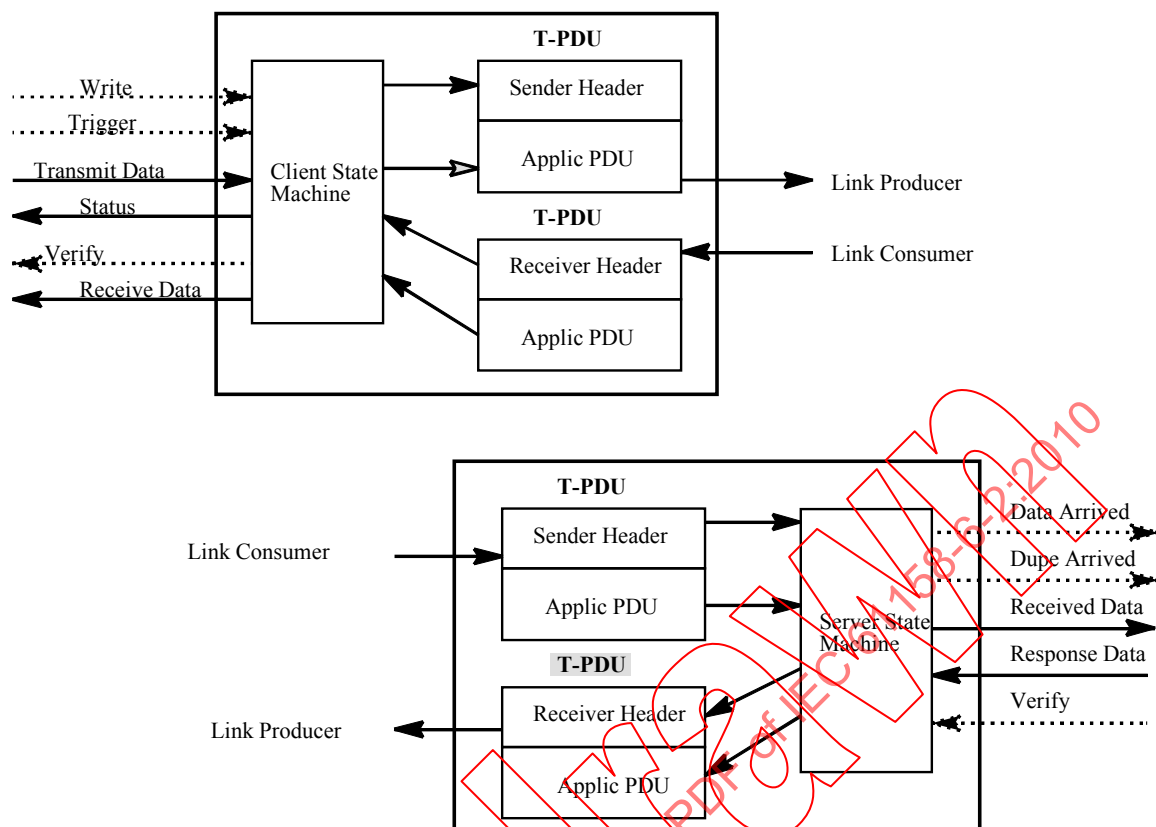


Figure 59 – Class 6 basic structure

9.3.9.5 Classes 4 to 6 general states

The defined states and their descriptions of the Class 4 Transport Sender are listed in Table 244.

Table 244 – Common states for transport classes 4 to 6

State	Description
Non-existent	The instance of Transport Class 4 to 6 does not exist.
Idle	The instance of Transport Class 4 to 6 has been created but not yet started.
Operational	The instance of Transport Class 4 to 6 has been created and started. NOTE This state has sub-states described in features particular to individual classes 4, 5 and 6.

9.3.9.6 Classes 4 to 6 general state transitions

Figure 60 shows the general behaviour of the transport classes. Each of the classes has some additional details added to the “Operational” state. This general behaviour shall **not** be repeated for each class in the interest of emphasizing the common behaviors and allowing those other descriptions to focus on the unique aspects of that class.

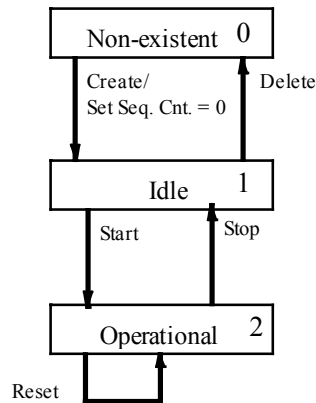


Figure 60 – Classes 4 to 6 general STD

9.3.9.7 Classes 4 to 6 general state event matrix

Table 245 defines the general state transitions of the transport classes.

Table 245 – Classes 4 to 6 general SEM

Event	State		
	Non-existent	Idle	Operational
Create	1) Seq. Cnt. = 0 2) Transition to Idle	Error	Error
Delete	Error	Transition to Non-existent	Error
Start	Error	Transition to Operational	Error
Stop	Error	No Action	Transition to Idle
Reset	Error	Error	Generally, stop waiting for outstanding events; details are transport class specific

9.3.9.8 Production triggers

This standard defines how these transport classes act as different trigger modes are specified when the Connection Manager creates an instance of the transport.

9.3.10 Transport state machines – class 4

9.3.10.1 Functions

The sequence count is set to 0 by the create service. It is incremented by the transport to distinguish a new Transport PDU from a retransmission of a previous one. It has a maximum value of 15. After that it rolls over to 0.

Transport class 4 is similar to having two class 2 transports, one in each direction, except that the class 4 uses two Network Connections instead of four. Transport class 4 can transfer application information in both directions. The delivery is acknowledged by the receiving transport. It uses a sequence number to identify which TPDU is being acknowledged. Retries are performed.

The main advantage of this class over using two Class 2 (Acknowledgement) transports is that class 4 uses fewer resources. The main advantage of class 4 over class 3 is that it does not block the application from sending a second request before the first response is received. This pipelining can improve the system performance in some circumstances. In addition, class 4 supports bi-directional, asynchronous data transmission, without imposing request-response type serialization.

This transport class notifies the application that the server application has responded to the transmitted packet. This transport class shall automatically send the next item in the queue when it receives an acknowledge that the previous one has been acknowledged. The application shall be notified if the network connection fails.

Transport class 4 uses two connections: the originator-to-target connection, which shall be point-to-point; and the return connection, which also shall be point-to-point.

The receiving transport initiates and sends the acknowledgement *after* the consuming node of the network connection has written the packet to the TPDU buffer, *after* the transport has read the packet's transport header, and *after* the transport has placed a copy of the data into a queue for the application. The receiver's acknowledgement signals the sending transport that the receiver-side TPDU receive buffer is ready to accept the next data packet.

The limit to the number of application requests which can be sent before an application response is received is determined by the design of the target application. It shall typically be limited by the amount of memory which the receiving transport can allocate for holding the outstanding messages. An error shall be returned to the sending transport when a transmission attempts to exceed the limit.

9.3.10.2 Transport class 4 sender

9.3.10.2.1 Functions

The class 4 Sender transport is responsible for:

- accepting the application's *write* that indicates the application has written a data packet to the transmit data queue;
 - writing the application data and the transport headers to the Sender TPDU transmit buffer for each packet that the application writes to it;
- NOTE The transport header for each packet shall include a sequence count.
- initializing the sequence count in the transport header (for the first packet transmitted) or incrementing the sequence count (for subsequent packets of the same transmission);
 - triggering the producing node of the network connection to produce the TPDU packet on the link and send it to the consuming node of the network connection;
 - performing retries if necessary when *triggered*;
 - accepting notification from the associated Receiver state machine that it has written the Receiver header (the other transport instance's verification) to the shared data area;
 - notifying the application that the application data has or has not been sent.

The following internal storage is assumed:

Current Sequence Number (SN)
Queue length
Try counter

9.3.10.2.2 States

The defined states and their descriptions of the Class 4 Transport Sender are listed in Table 246.

Table 246 – Class 4 transport sender states

State	Description
Ready	The instance of Transport Class 4 has been created and started. Queue is empty.
Waiting	The instance of Transport Class 4 has been created and started. Queue is not empty.

NOTE These two states are sub-states of the Running state described as the common feature for classes 4 to 6.

9.3.10.2.3 State transitions

State transitions of the class 4 sender transport are shown in Figure 61.

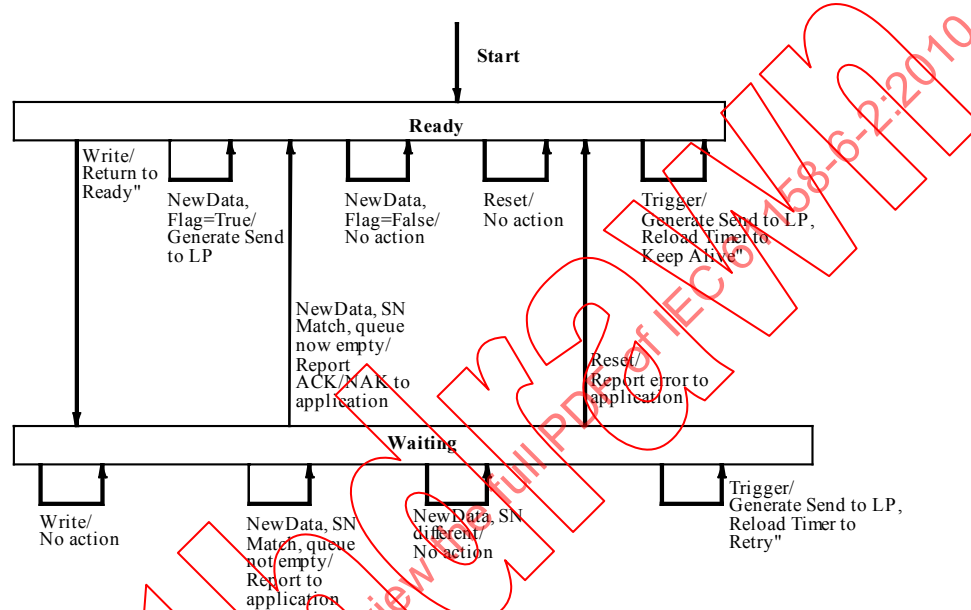


Figure 61 – Class 4 sender STD

NOTE 1 This STD is incomplete. It shows the most common transitions. See Table 247 for the full specification of all events in all conditions.

The Command and Sequence Number columns in Table 247 refer to the values of those fields of the *To_Send* data item. In the action descriptions, the phrase “set CMD” refers to setting the value of the Command field of the Sender Header in the TPDU which shall be delivered to the Link Producer at the next *Send* event.

On all occurrences of the *New Data* event, the contents of the *From_Recv* data item are copied to the Receiver Header in the TPDU.

NOTE 2 This action is not listed in Table 247.

9.3.10.2.4 State event matrix

State transitions of the class 4 sender transport are shown in Table 247.

Table 247 – Class 4 sender SEM

Event	Command	Sequence Number	State	
			Ready	Waiting
Reset	Don't care	Don't care	Do "return to Ready processing"	Take the application data off the queue, place error in Status queue, generate Status Updated event to application, do "return to Ready processing"
Write	Don't care	Don't care	Do "return to Ready processing"	Ignore (Shall be recognized as part of "return to Ready processing")
NewData	ACK	Different	(Not expecting anything). Do "Flag processing".	(Assume the other transport is ACKing something old) Do "Flag processing".
		Same		Take the application data off the queue, place Success in Status queue, generate Status Updated event to application, do "return to Ready processing"
	NAK	Different		(Assume the other transport is NAKing something old) Do "Flag processing".
		Same		Take the application data off the queue, place error in Status queue, generate Status Updated event to application, do "return to Ready processing"
	Other	Don't care		(Assume other end is confused.) Do "Flag processing".
Trigger (Retry or Keep alive timer)	Don't care	Don't care	Generate Send event to Link Producer (which re-sends the <i>Null</i> message to keep connection alive), reload timer with "keep alive" value.	Decrement Try counter, if zero: Take the application data off the queue, report error to application, generate Can't Send event to application, do "return to Ready processing" else: Generate Send event to Link Producer (which re-sends the <i>Only</i> message), reload timer with retry value.

The "Return to Ready Processing" includes the following:

- If the queue is now empty:
 - Increment Sequence number and place in Sender Header of TPDU
 - Set CMD to *Null*
 - Load timer with "keep alive" value
 - Do the "Flag Processing" (described below)
 - Go to Ready state
- If the queue is not empty:
 - Increment Sequence number and place in Sender Header of TPDU
 - Set CMD to *Only*
 - Set Try counter
 - Copy message to TPDU buffer
 - Generate Send event to Link Producer
 - Load timer with "retry" value
 - Go to Waiting state

The "Flag Processing" is defined as: Test the value of the Flag data item from the Receiver state machine. If the Flag has a value of True then generate the Send event to the Link Producer. If the Flag has a value of False, do nothing.

9.3.10.3 Transport class 4 receiver

9.3.10.3.1 Functions

In the link to application direction, a class 4 Receiver transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side TPDU receive buffer;
- locating and reading the transport headers of each packet that the consuming node of the network connection writes to the server-side TPDU receive buffer;
- performing the actions specified in the state event matrix for each packet which arrives;
- notifying the server application when it places new data into the received data queue;
- deliver the appropriate data (*To_Send*, *From_Recv*, and *Flag*) to the associated Sender transport state machine with the *NewData* event.

The Receiver Bi-directional Point-to-Point Non-Blocking transport has two states:

Initial
Ready

The basic approach is to start in the *Initial* state and transition to the *Ready* state when it receives something that indicates the receiver is synchronized with the sender. It remains in the *Ready* state until the *Reset* service is applied.

9.3.10.3.2 States

The defined states and their descriptions of the Class 4 Transport Receiver are listed in Table 248 below :

Table 248 – Class 4 transport receiver states

State	Description
Ready	The instance of Transport Class 4 has been created and started. Queue is empty.
Waiting	The instance of Transport Class 4 has been created and started. Queue is not empty.

NOTE These two states are sub-states of the Running state described as the common feature for classes 4 to 6.

9.3.10.3.3 State transitions

State transitions of the class 4 receiver transport are shown in Figure 62.

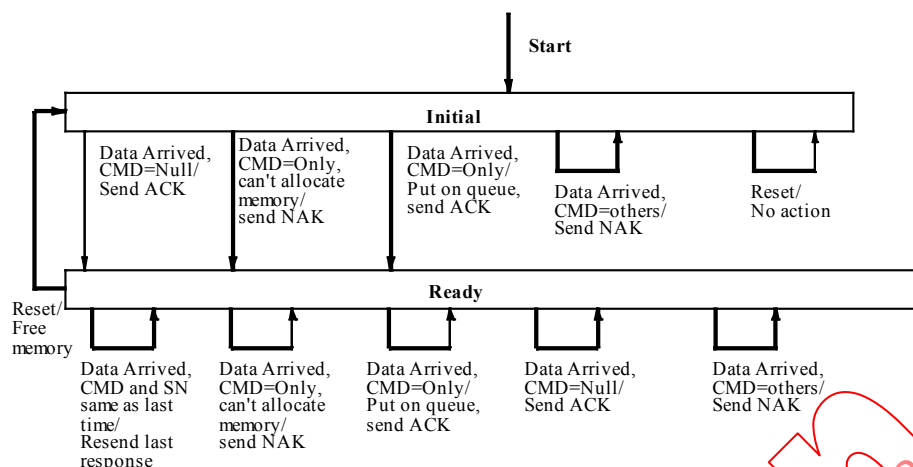


Figure 62 – Class 4 receiver STD

The following events can happen. Some of these are labeled on the Data Flow Diagram as the Packet Arrived event; I have taken the liberty to also factor in what arrived and if it has the same or different Sequence Number (SN):

- Data Arrived, CMD = *Null*;
- Data Arrived, CMD = *Only*;
- Data Arrived, CMD is *Illegal*;
- Reset.

NOTE 1 There is **no** timer event used by the Receiver state machine. (If there is a communications problem, the Sender state machine should time out. The application should have a timer to indicate problems with the other application).

On all occurrences of the *Data Arrived* event, the contents of the Receiver Header are copied to the *To_Send* data item.

NOTE 2 This action is not listed in Table 249.

The phrase “Set CMD” used in Table 249 means “set the Command field of the *From_Recv* data item”. When the value of CMD or SN is used to make a decision, it refers to the value of the Command or Sequence Number fields of the Sender Header in the TPDU received from the Link Consumer which caused the Data Arrived event.

The following internal storage is assumed:

Previous SN

Previous CMD received

9.3.10.3.4 State event matrix

State transitions of the class 4 receiver transport are shown in Table 249.

Table 249 – Class 4 receiver SEM

Event	State	
	Initial	Ready
Reset	No action	Set the Command field of the <i>From_Recv</i> data item to <i>Not_Init</i> and the Sequence Number of the <i>From_Recv</i> data item to zero. Go to Initial.
Data Arrived, CMD=Null, SN different	Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate NewData event, go to Ready.	Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate NewData event, stay in Ready.
Data Arrived, CMD=Null, SN same		If previous CMD was <i>Null</i> , then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Set Flag to False, generate NewData event, stay in Ready. (There is no need to rush out an ACK to a <i>Null</i> , let the sender's refresh send it out.) else: (Assume other transport made an error by sending a different CMD, but used the same SN). Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to False, generate NewData event, stay in Ready.
Data Arrived, CMD=Only, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to True, generate NewData event, go to Ready. else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to True, generate NewData event, stay in Ready. else: Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, stay in Ready.
Data Arrived, CMD=Only, SN same		If previous CMD was <i>Only</i> , then: Set Flag to True, generate NewData event, stay in Ready. (This is a retry, assume SN and CMD are OK.) (Flag is TRUE to speed recovery from noise, at the possible expense of some additional link transmit time.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.
Data Arrived, CMD=Illegal	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate NewData event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate NewData event, stay in Ready.

9.3.11 Transport state machines – class 5

9.3.11.1 Functions

Possible uses of transport class 5 include exchanging messages and responses up to 65 536 octets between two applications.

The sequence count is set to 0 by the create service. It is incremented by the transport to distinguish a new Transport PDU from a retransmission of a previous one. It has a maximum value of 15. After that it rolls over to 0 limiting the transport to fewer than 16 outstanding messages.

Transport class 5 is similar to class 4, except that class 5 adds Fragmentation and Reassembly at the transport layer. Transport class 5 can transfer application information in

both directions. The delivery of each fragment is acknowledged by the receiving transport. A sequence number identifies which TPDU is being acknowledged. Retries are performed.

This transport class notifies the application that the server application has responded to the transmitted packet. This transport class shall automatically send the next item in the queue when it receives an acknowledgement that the previous one has been acknowledged. The application shall be notified if the network connection fails.

Transport Class 5 uses two connections: the originator-to-target connection, which shall be point-to-point, and the return connection, which also shall be point-to-point.

The receiving transport initiates and sends the acknowledgement *after* the consuming node of the network connection has written the packet to the TPDU buffer, *after* the transport has read the packet's transport header, and *after* the transport has placed a copy of the data into a queue for the application. The receiver's acknowledgement signals the sending transport that the receiver-side TPDU receive buffer is ready to accept the next data packet.

The limit to the number of application requests which can be sent before an application response is received is determined by the design of the target application. It shall typically be limited by the amount of memory. An error shall be returned when a transmission attempts to exceed the limit.

9.3.11.2 Class 5 sequence diagrams

9.3.11.2.1 Typical sequences

Some typical sequences are illustrated to aid understanding of the operation of this transport class. A formal specification of this transport class is given in IEC 61158-5-2:2010.

NOTE The following sequences are identical for classes 4 and class 5. They are described under the class 4 specification and are not repeated here.

- Typical Message Exchange Sequence (see 9.3.9.3.2)
- Typical Sequence with Messages overwriting each other (see 9.3.9.3.3)
- Typical Queued Sequence (see 9.3.9.3.4)
- Typical Sequence with retries (see 9.3.9.3.5)
- Typical Idle Sequence (see 9.3.9.3.6).

9.3.11.2.2 Typical three fragment sequence

Figure 63 shows a typical sequence where the application data is too large to fit into one TPDU. In the example shown, the application data is split into three fragments. In general, there could be many *Middle* TPDU's. Each *Middle* TPDU increments the Sequence Number in the Sender Header. The *Last* TPDU shall also increment the Sequence Number.

At the receiver, the *First* allocates the memory (and supplies the data for the first fragment) and the *Last* indicates it is time to notify the receiving application that there is some data for it to process (as well as supplying the data for the last fragment).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

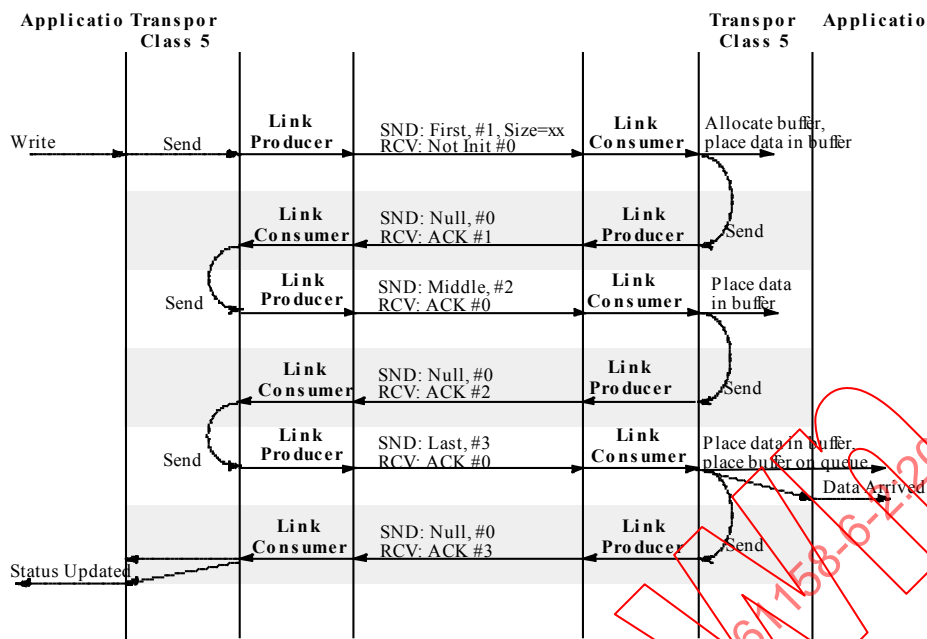


Figure 63 – Sequence diagram of three fragments using transport class 5

9.3.11.2.3 Typical fragmented sequence with retries

Figure 64 shows the processing when a transmission error causes something to be lost. In this example, two transmissions are lost, one in each direction.

NOTE The processing is very similar to class 4 (see 9.3.9.3.5).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

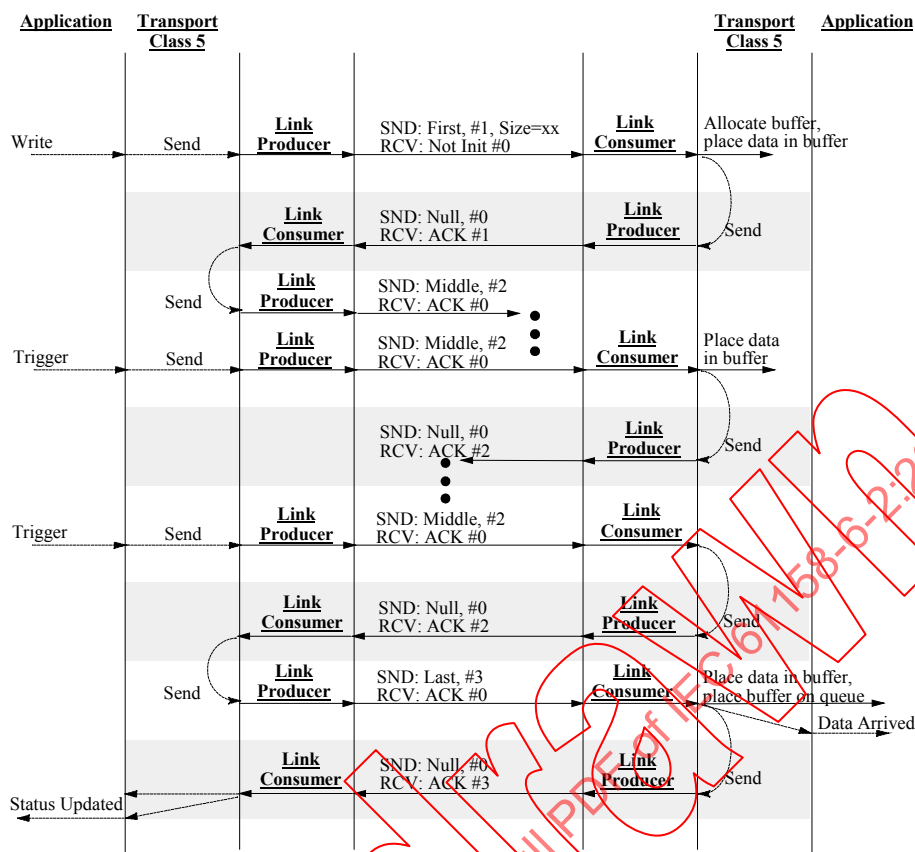


Figure 64 – Sequence diagram of fragmentation with retries using transport class 5

9.3.11.2.4 Typical two fragment sequence

Figure 65 shows a special case, where there are only two fragments. In this case, there is no *Middle* TPDU. Again, the *First* allocates the memory (and supplies the data for the first fragment) and the *Last* indicates it is time to notify the receiving application that there is some data for it to process (as well as supplying the data for the last fragment).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

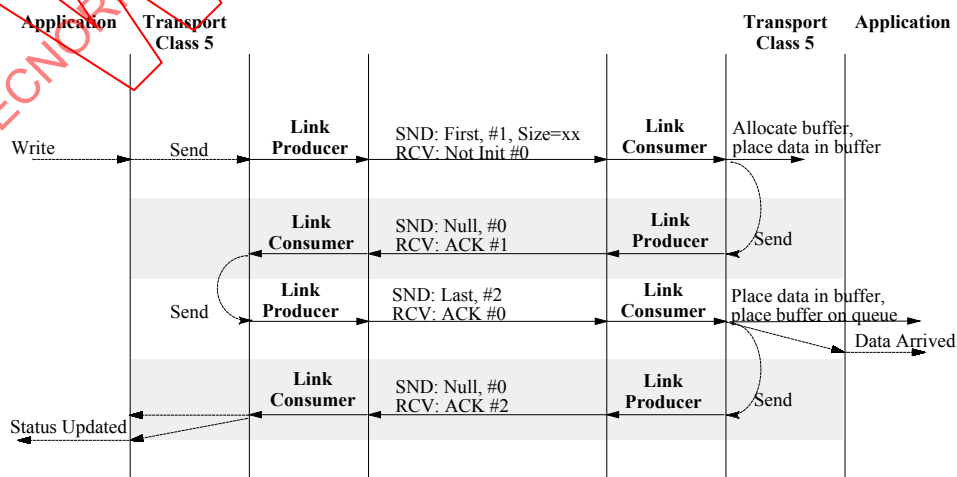


Figure 65 – Sequence diagram of two fragments using transport class 5

Figure 66 shows a receiver recovering when the sender has been reset in the middle of a fragmented sequence. When the receiver gets a *First* or *Only*, it assumes that any message which did not receive a *Last* has been aborted.

[illegible]

9.3.11.3 Transport class 5 sender

The Class 5 Sender transport is responsible for:

- The following internal storage is assumed:

Queue length;

Try counter.

9.3.11.3.2 States

The defined states and their descriptions of the Class 5 Transport Sender are listed in Table 250.

Table 250 – Class 5 transport sender states

State	Description
Ready	The instance of Transport Class 5 has been created and started. Queue is empty.
Waiting	The instance of Transport Class 5 has been created and started. Queue is not empty.

NOTE These two states are sub-states of the Running state described as the common feature for classes 4 to 6.

9.3.11.3.3 State transitions

State transitions of the class 5 sender transport are shown in Figure 67.

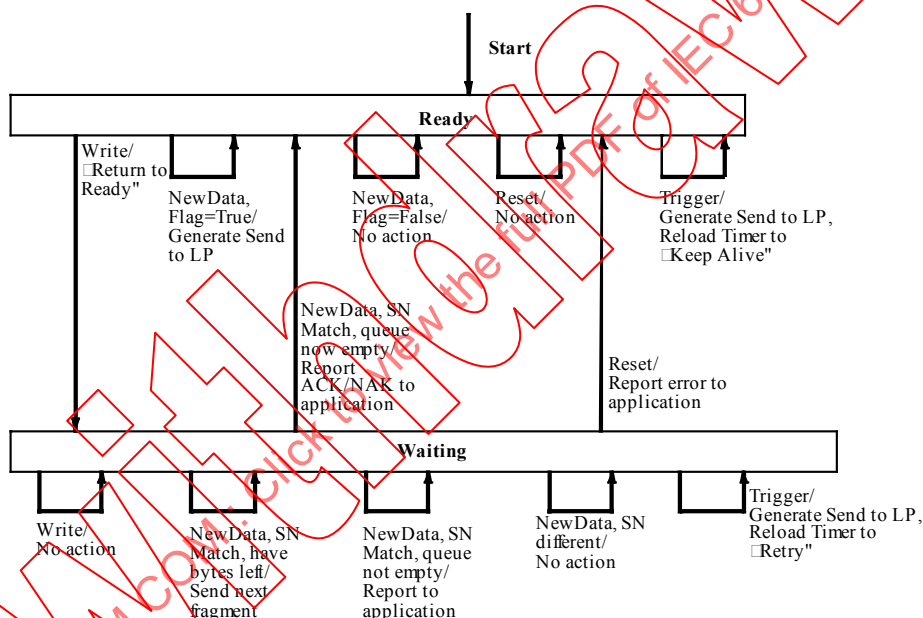


Figure 67 – Class 5 sender STD

NOTE 1 This STD is incomplete. It shows some common transitions. See Table 251 for the full specification of all events in all conditions.

The Command and Seq. Number columns in Table 251 refer to the values of those fields of the *To_Send* data item. In the action descriptions, the phrase “set CMD” refers to setting the value of the Command field of the Sender Header in the TPDU which shall be delivered to the Link Producer at the next *Send* event. Also, the phrase “increment sequence number” refers to setting the value of the Sequence Number field of the Sender Header in the TPDU which shall be delivered to the Link Producer at the next *Send* event.

On all occurrences of the *New Data* event, the contents of the *From_Recv* data item are copied to the Receiver Header in the TPDU.

NOTE 2 This action is not listed in Table 251.

9.3.11.3.4 State event matrix

State transitions of the class 5 sender transport are shown in Table 251.

Table 251 – Class 5 sender SEM

Event	Command	Sequence number	State	
			Ready	Waiting
Reset	n/a	n/a	Do “return to Ready processing”	Take the application data off the queue, place error in Status queue, generate Status Updated event to application, do “return to Ready processing”
Write	n/a	n/a	Do “return to Ready processing”	Ignore (Shall be recognized as part of “return to Ready processing”)
NewData	ACK	Different	(Not expecting anything). Do “Flag processing”.	(Assume the other transport is ACKing something old) Do “Flag processing”.
		Same		If the whole application message has been sent, then take the application data off the queue, place Success in Status queue, generate Status Updated event to application, do “return to Ready processing” else, if the remainder fits in one fragment, then place the remainder in the Link Producer buffer, increment the sequence count, set the CMD to <i>Last</i> , and generate the Send event to the Link Producer else, place the next fragment in the Link Producer buffer, increment the sequence count, set the CMD to <i>Middle</i> , and generate the Send event to the Link Producer and stay in Waiting state.
	NAK	Different		(Assume the other transport is NAKing something old) Do “Flag processing”.
		Same		Take the application data off the queue, place error in Status queue, generate Status Updated event to application, do “return to Ready processing”
	Other	n/a		(Assume other end is confused.) Do “Flag processing”.
Trigger (Retry or Keep alive timer)	n/a	n/a	Generate Send event to Link Producer (which re-sends the <i>Null</i> message to keep connection alive), reload timer with “keep alive” value.	Decrement Try counter, if zero: Take the application data off the queue, report error to application, generate Can't Send event to application, do “return to Ready processing” else: Generate Send event to Link Producer (which re-sends the previous message), reload timer with retry value.

The “Return to Ready Processing” includes the following:

If the queue is now empty:

- Increment Sequence number
- Set CMD to *Null*
- Load timer with “Keep alive” value
- Do the “Flag Processing” (described below)
- Go to Ready state

If the queue is not empty:

- Increment Sequence number

Test if the message shall fit in one fragment; if it does:

- Set CMD to Only
- Copy message to TPDU buffer

else

- Set CMD to First
- Calculate (total size / 256 octets) – 1; put into Sender Header
- Copy first fragment to TPDU buffer

Set Try counter

Load timer with “Retry” value

Generate Send event to Link Producer

Go to Waiting state

The “Flag Processing” is defined as: Test the value of the Flag data item from the Receiver state machine. If the Flag has a value of True then generate the Send event to the Link Producer. If the Flag has a value of False, do nothing.

9.3.11.4 Transport class 5 receiver

9.3.11.4.1 Functions

In the link to application direction, a Class 5 Receiver transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side TPDU receive buffer;
- locating and reading the transport headers of each packet that the consuming node of the network connection writes to the server-side TPDU receive buffer;
- performing the actions specified in the state event matrix for each packet which arrives;
- notifying the server application when it places new data into the received data queue;
- deliver the appropriate data and flags to the associated Sender transport.

The Receiver Bi-directional Point-to-Point Non-Blocking transport has three states:

Initial;

Assembling;

Ready.

The following internal storage is assumed:

Previous Sequence Number (SN);

Previous Command (CMD).

The basic approach is to start in the *Initial* state and transition to either the *Assembling* or the *Ready* state when it receives something that indicates the receiver is synchronized with the sender. It then stays in the *Ready* and *Assembling* states until the *Reset* service is applied. It goes into the *Assembling* state after receiving a *First* TPDU and remains there until getting the *Last* TPDU (unless it thinks the sender has aborted sending the current T-SDU).

9.3.11.4.2 States

The defined states and their descriptions of the Class 5 Transport Receiver are listed in Table 252 below :

Table 252 – Class 5 transport receiver states

State	Description
Initial	The instance of Transport Class 5 has been created and started.
Assembling	The instance of Transport Class 5 has been created and started. Segmented data is being assembled.
Ready	The instance of Transport Class 5 has been created and started. Segmented data has been assembled.

NOTE These three states are sub-states of the Running state described as the common feature for classes 4 to 6.

9.3.11.4.3 State transitions

State transitions of the class 5 receiver transport is shown in Figure 68.

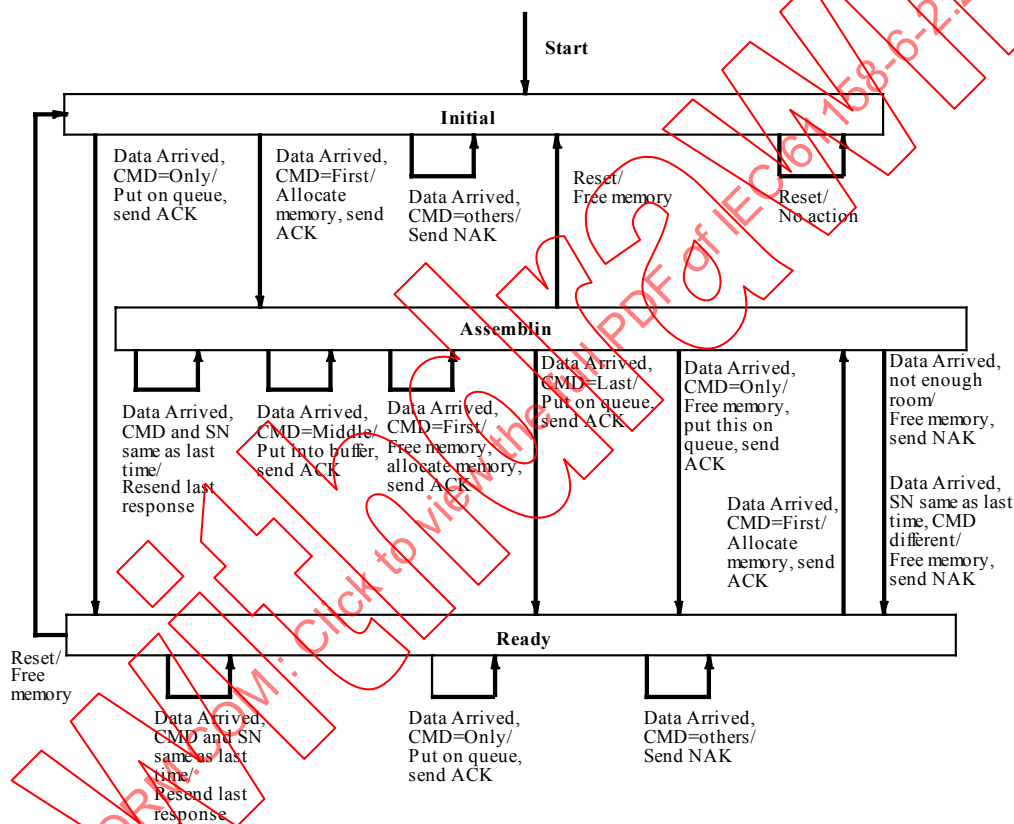


Figure 68 – Class 5 receiver STD

The following events can happen.

- *Reset* Event;
- Data Arrived Event, CMD= Null;
- Data Arrived Event, CMD= Only;
- Data Arrived Event, CMD= First;
- Data Arrived Event, CMD= Middle;
- Data Arrived Event, CMD= Last;
- *Data Arrived* Event, CMD= Illegal command.

NOTE There is **no** timer event used by the Receiver state machine. (If there is a communications problem, the Sender state machine should time out. The application should have a timer to indicate problems with the other application.)

The following internal storage is assumed:

- Previous SN;
- Previous CMD received;
- Buffer Pointer;
- Room Left.

9.3.11.4.4 State event matrix

State transitions of the class 5 receiver transport is shown in Table 253.

IECNORM.COM: Click to view the full PDF of IEC 61158-6-2:2010

Withdawn

Table 253 – Class 5 receiver SEM

Event	State		
	Initial	Ready	Assembling
Reset	No action.	Go to Initial.	Free memory previously allocated. Go to Initial.
Data Arrived, CMD=Null, SN different	Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate NewData event, go to Ready.	Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate NewData event, stay in Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to False, generate NewData event, stay in Assembling.
Data Arrived, CMD=Null, SN same		If previous CMD was <i>Null</i>, then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Set Flag to False, generate NewData event, stay in Ready. else: (Assume other transport made an error by sending a different CMD, but used the same SN). Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to False, generate NewData event, stay in Ready.	If previous CMD was <i>Null</i>, then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Set Flag to False, generate NewData event, stay in Assembling. else: (Assume other transport made an error by sending a different CMD, but used the same SN). Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to False, generate NewData event, go to Ready.
Data Arrived, CMD=Only, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i>, set CMD to ACK, set Flag to True, generate NewData event, stay in Initial else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i>, set CMD to ACK, set Flag to True, generate NewData event, stay in Ready else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), set Flag to True, generate NewData event, stay in Ready.	Free memory previously allocated. Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Arrived event, copy SN to <i>From_Recv</i>, set CMD to ACK, set Flag to True, generate NewData event, go to Ready. else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.
Data Arrived, CMD=Only, SN same		If previous CMD was <i>Only</i>, then: Set Flag to True, generate NewData event, stay in Ready. (This is a retry, assume SN and CMD are OK.) (Flag is TRUE to speed recovery from noise, at the possible expense of some additional link transmit time.) else: Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Assembling. (The sender changed the CMD but not the SN.)

Event	State		
	Initial	Ready	Assembling
Data Arrived, CMD=First, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i>, initialize Buffer Pointer and Room Left, set CMD to ACK, set Flag to True, generate NewData event, go to Assembling state. else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i>, initialize Buffer Pointer and Room Left, set CMD to ACK, set Flag to True, generate NewData event, go to Assembling state. else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), set Flag to True, generate NewData event, stay in Ready.	Free memory previously allocated. Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i>, initialize Buffer Pointer and Room Left, set CMD to ACK, set Flag to True, generate NewData event, stay in Assembling else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.
Data Arrived, CMD=First, SN same		If previous CMD was <i>First</i>, then: Set Flag to True, generate NewData event, stay in Ready. (This is a retry, previous attempt errored, assume SN and CMD are OK.) else: Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	If previous CMD was <i>First</i>, then: Set Flag to True, generate NewData event, stay in Assembling. (This is a retry, assume SN and CMD are OK.) else: Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Assembling.
Data Arrived, CMD=Middle, SN different	Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Initial.	Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, copy SN to <i>From_Recv</i>, update Buffer Pointer and Room Left, set CMD to ACK, set Flag to True, generate NewData event, stay in Assembling. else: Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.
Data Arrived, CMD=Middle, SN same			If previous CMD was <i>Middle</i>, then: Set Flag to False, generate NewData event, stay in Assembling. (This is a retry, assume SN and CMD are OK.) else: Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), set Flag to True, generate NewData event, go to Ready.

Event	State		
	Initial	Ready	Assembling
Data Arrived, CMD=Last, SN different	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, put buffer on Received queue, generate Arrived event, copy SN to <i>From_Recv</i> , set CMD to ACK, set Flag to True, generate NewData event, go to Ready. else: Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (no memory), set Flag to True, generate NewData event, go to Ready.
Data Arrived, CMD=Last, SN same		If previous CMD was Last, then: Set Flag to True, generate NewData event, stay in Ready. (This is a retry, assume SN and CMD are OK.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Ready.	If previous CMD was Last, then: Set Flag to False, generate NewData event, go to Ready. (This is a retry, assume SN and CMD are OK.) NOTE This shall happen: If the previous CMD is Last, then we shall be in the Ready state.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), set Flag to True, generate NewData event, stay in Assembling.
Data Arrived, CMD=Illegal	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate NewData event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate NewData event, stay in Ready.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), set Flag to True, generate NewData event, stay in Assembling.

9.3.12 Transport state machines – class 6

9.3.12.1 Functions

Possible uses of transport class 6 include delivering event notification messages (up to 65 536 octets) to multiple servers.

Transport class 6 is similar to class 3. Transport class 6 can transfer application information in both directions. Class 6 supports multicasting in the Client to Server direction. Class 6 adds Fragmentation and Reassembly at the transport layer, but only in the Client to Server (multicast) direction.

The delivery of each fragment is acknowledged by the receiving transport. A sequence number identifies the TPDU being acknowledged. Retries are performed.

The sequence count is set to 0 by the create service. It is incremented by the transport to distinguish a new Transport PDU from a retransmission of a previous one. It has a maximum value of 15. After that it rolls over to 0.

Class 6 is NOT a superset of class 5. Class 6 is blocking; that is, the next application request is not delivered until the current one is acknowledged. Functionally, class 6 can be thought of as a fragmenting version of class 3, rather than as a multicasting version of class 5. However, class 6 reuses the transport header format from class 5, instead of the class 3 transport header.

This transport class notifies the application that the server application has responded to the transmitted packet. The application shall be notified if the network connection fails.

Transport class 6 uses multiple network connections: the Client-to-Server connection, which may be a multipoint connection; and the return connections (one per Server), which shall be point-to-point.

One significant difference between class 3 and class 6, which results from class 6 doing fragmentation: the “write” event causes the first (or only) fragment to be transmitted; the “trigger” event is only a timer function which is used to signal retries or keep-alive transmissions.

9.3.12.2 Class 6 data flow diagram

Figure 69 shows the data flow diagram for class 6.

NOTE There is a separate T-SDU and TPDU in the Client to Server direction. The client places the fragments into the TPDU. The server assembles them from the TPDU into the T-SDU. Since there is no fragmentation in the Server to Client direction, the Application places the data directly into the Server's outgoing TPDU (as is done in class 3).

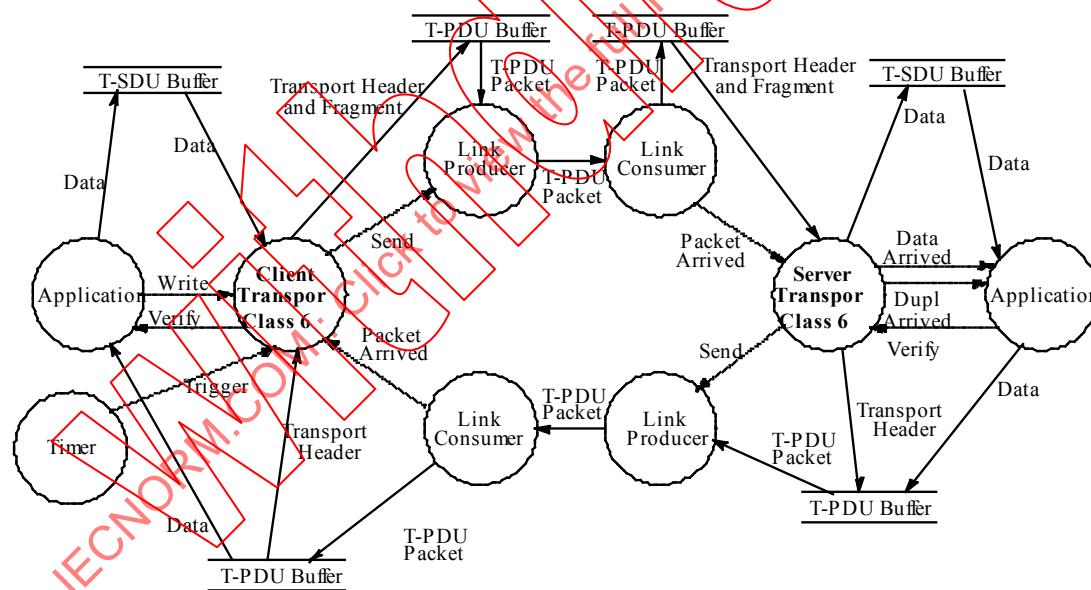


Figure 69 – Data flow diagram for transport class 6

9.3.12.3 Class 5 features missing in class 6

9.3.12.3.1 Function

It may be noted that the following two features of class 5 are not supported in class 6:

- the ability to have multiple outstanding requests queued between the receiving transport and its application;
- the support of fragmentation and reassembly in both directions.

This is a protocol design decision to exclude these features for the following reasoning. It is preferable that the class 6 transport header will not change size during the lifetime of a

connection (due to the complexity of notifying all receivers of the change in header size). In the system, the multicasting receivers can enter and leave the relationship at will. Therefore, the number of multicasting receivers can change during the lifetime of the connection. Therefore, the size of the header shall be independent of the number of receivers. Since it is desirable to have no artificial limit to the number of receivers the decision was made to have **one** header field which all receivers use. This implies that all receivers shall be in the same state, which is maintained by having them all synchronized with the sender.

9.3.12.3.2 Blocking versus non-blocking

Assume there are several receivers of the multicast data. They are sending application responses to application requests to the multicasting sender. These application responses need to be acknowledged so that the transport can tell if it shall retry sending them or if it can now free up the resources used by that response. If the application responders are not synchronized with each other, they can be responding to different application requests. Therefore, each needs a field so that the application requester's transport can tell each of them about the successful delivery of the application response.

The design decision was not to have multiple fields, therefore all use the same field to decide if a retry is needed or if the resources can be released. Therefore, the transport delivery is synchronized with the application (the same as class 3).

9.3.12.3.3 Fragmentation in both directions

This was excluded by similar reasoning to the "Blocking vs. Non-Blocking" discussion. If the application responses from different responders require a different number of fragments, it would require the multicaster to send different transport headers to each of them.

9.3.12.4 Class 6 sequence diagrams

9.3.12.4.1 Typical sequences

Some typical sequences unique to class 6 are illustrated to aid understanding of the operation of this transport class. A formal specification of this transport class is given in IEC 61158-5-2:2010.

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

9.3.12.4.2 Typical message exchange sequence

Figure 70 shows the sequence in which actions take place during a messaging request/response exchange using transport class 6. The unshaded areas in Figure 70 indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction. Each transmission shows the Command and Sequence Number used for each header.

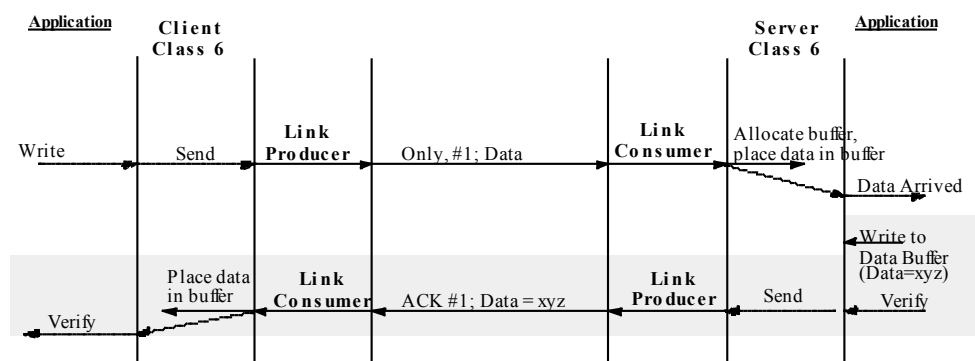


Figure 70 – Sequence diagram of message exchange using transport class 6

9.3.12.4.3 Typical message exchange sequence with retries

Retries are performed when a transport does not receive an Receiver header with the same sequence count as its current Sender header in the timeout period. The TPDU is not changed for a retry. Figure 71 shows two retry situations: where an *Only* command fails to be delivered and where an expected ACK fails to be delivered. This processing is very similar to class 3. The Trigger event is generated by the retry timer.

NOTE See 9.3.12.4.7 for a typical sequence with retries when the data is fragmented.

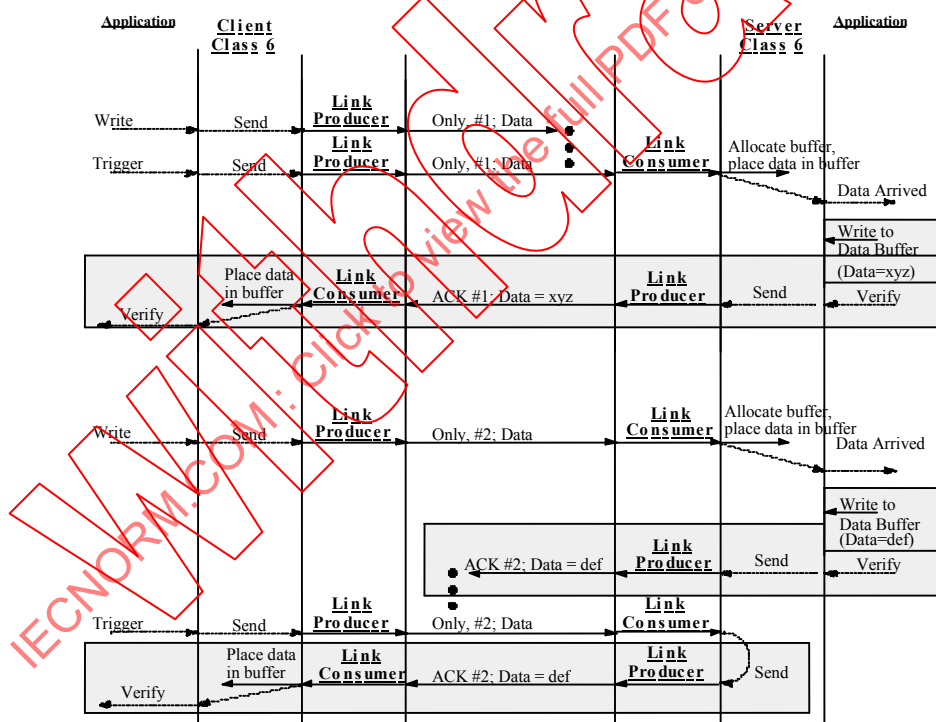


Figure 71 – Sequence diagram of retries using transport class 6

9.3.12.4.4 Typical idle sequence

When a class 6 Client transport has nothing to send which shall impact the other state machine, it does not need to send anything. However, it is necessary to send some small amount of traffic to keep the connection alive. The Trigger event shall be used, with the timer set to a "keep alive" value. Figure 72 shows the *Null* command used to keep the connection alive, using the Trigger event.

The Server does not have a timer, therefore the *Null* command shall be acknowledged by the server to keep the point to point network connection open. However, the client transport does

not need to wait for the acknowledgement if the application gives it a Write event (see 9.3.12.4.5).

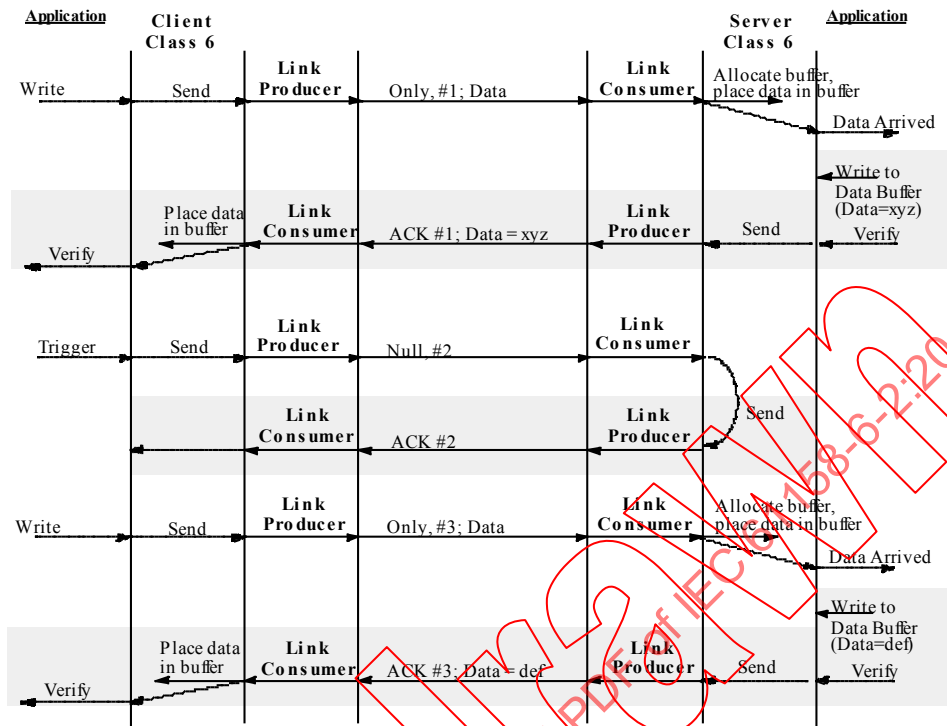


Figure 72 – Sequence diagram of idle traffic using transport class 6

9.3.12.4.5 Typical sequence with overwrite

It is permissible for a transport to send two data packets in a row. This implies that the second one might overwrite the first one in an intermediate module. This is desirable, since the second packet contains information that is newer than the first one and the data in the first one is no longer needed. There are two situations where this can occur when using Transport class 6.

In Figure 72 the transport on the left had nothing to send when it was triggered by the keep-alive timer. Therefore, it sent the *Null #2*. But when the application on the left did a *WRITE*, the transport immediately sent the *Only #3*. The previous example shows both of them arriving separately.

Figure 73 shows the left side's *Only #3* overwriting the *Null #2*.

Figure 74 shows a different situation where an overwrite can occur. The left side sends a *Null #2* as the result of the keep-alive *Trigger* event. But, before the *ACK #2* can be delivered, the right side receives the *Only #3* from the left side. The right side responded with the newer *ACK #3*, which overwrites the *ACK #2*.

The same situation shall occur if a *First* is used instead of the *Only*.

It is not necessary to wait for the ACK to a *Null* before sending an *Only* or a *First*.

When sending an *Only* or a *First*, it shall have a different sequence number than the *Null*, so sender knows which is being ACK'd by the other side.

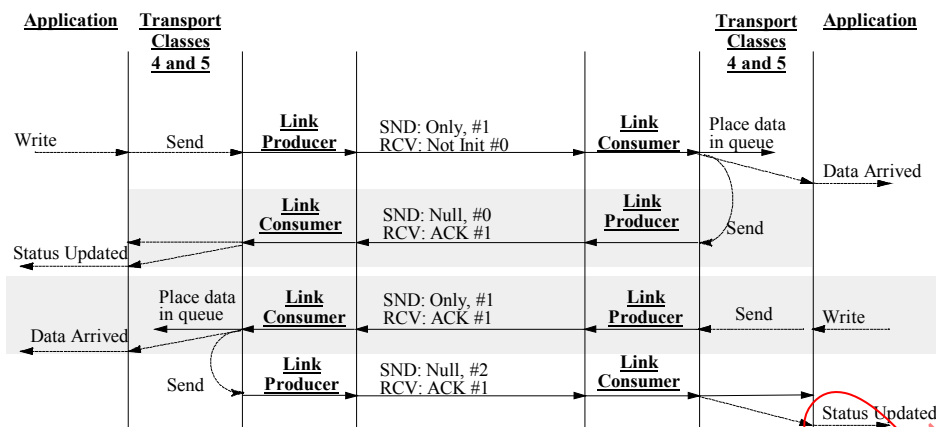


Figure 73 – Sequence diagram of request overwriting null

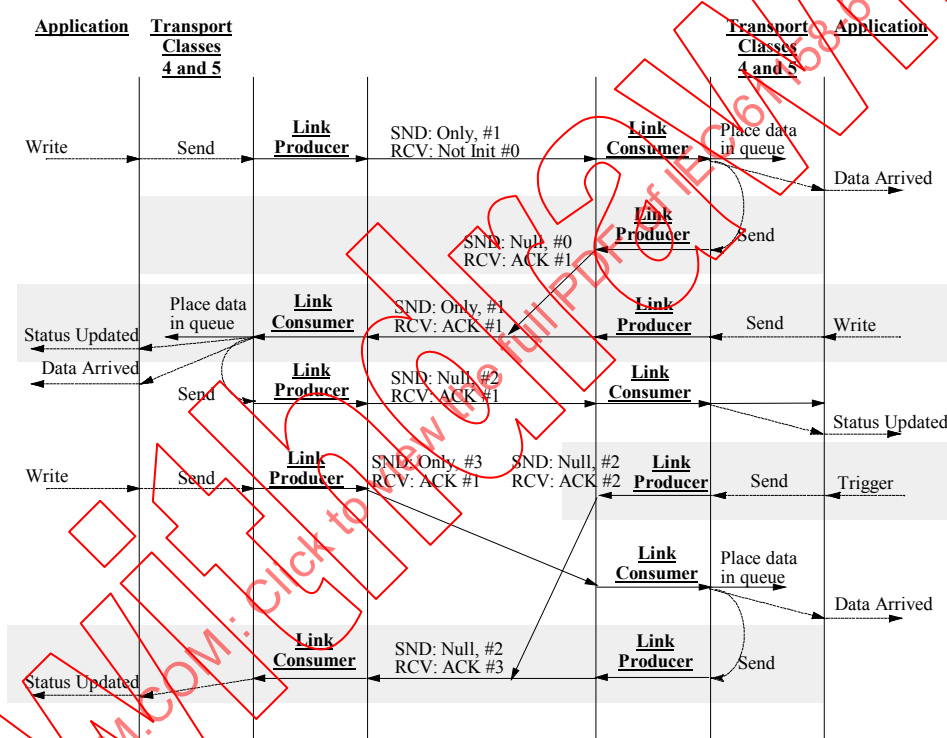


Figure 74 – Sequence diagram of response overwriting ACK of null

9.3.12.4.6 Typical three fragment sequence

Figure 75 shows a typical sequence where the application data is too large to fit into one TPDU. In the example shown, the application data is split into three fragments. In general, there could be many *Middle* TPDU. Each *Middle* TPDU increments the Sequence Number in the Sender Header. The *Last* TPDU shall also increment the Sequence Number.

At the receiver, the *First* allocates the memory (and supplies the data for the first fragment) and the *Last* indicates it is time to notify the receiving application that there is some data for it to process (as well as supplying the data for the last fragment).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

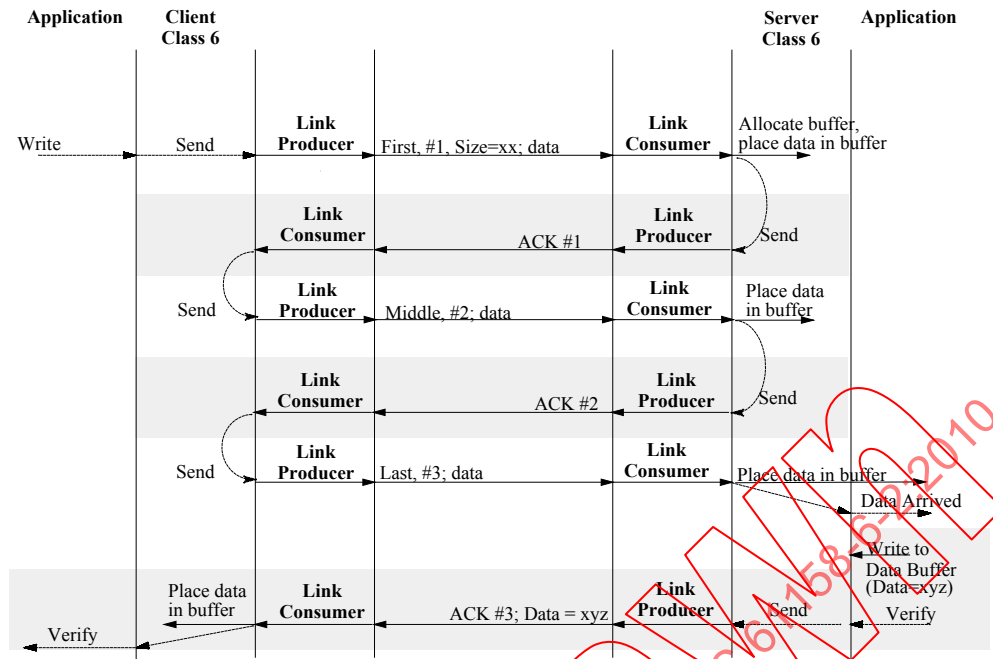


Figure 75 – Sequence diagram of three fragments using transport class 6

9.3.12.4.7 Typical fragmented sequence with retries

Figure 76 shows the processing when a transmission error causes something to be lost. In this example, two transmissions are lost, one in each direction. The processing is very similar to Class 5.

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

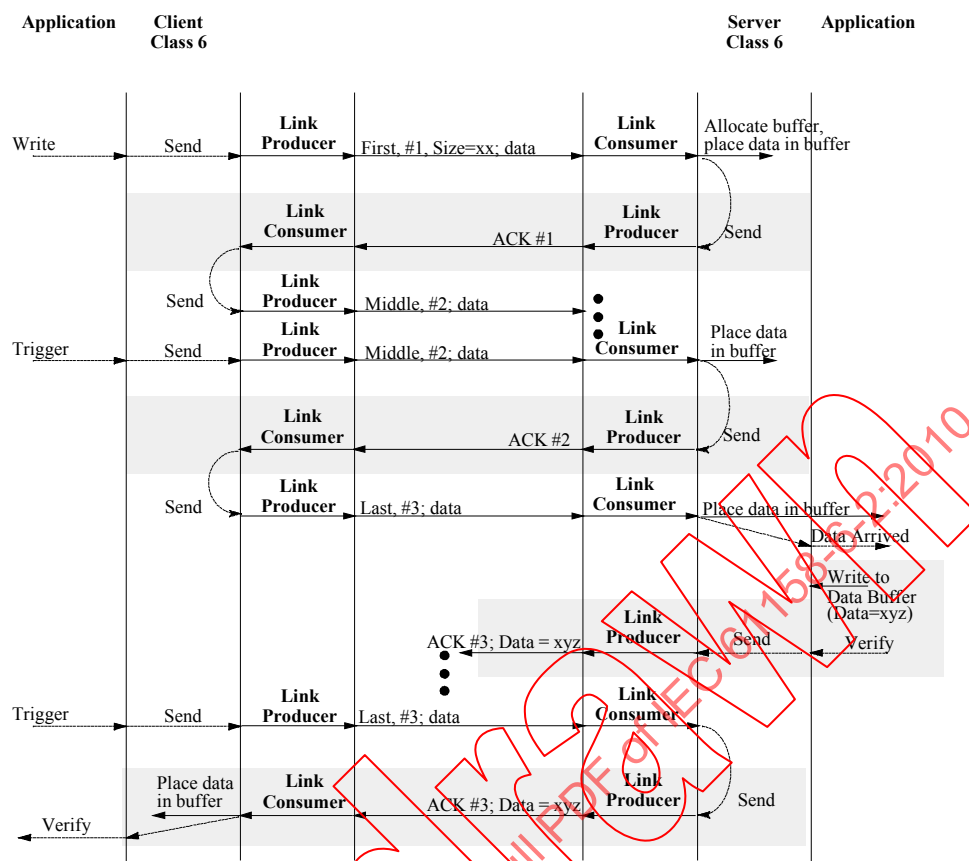


Figure 76 – Sequence diagram of fragmentation with retries using transport class 6

9.3.12.4.8 Typical two fragment sequence

Figure 77 shows a special case, where there are only two fragments. In this case, there is no *Middle* TPDU. Again, the *First* allocates the memory (and supplies the data for the first fragment) and the *Last* indicates it is time to notify the receiving application that there is some data for it to process (as well as supplying the data for the last fragment).

The unshaded areas in the figures indicate data transmission in one direction, and the shaded areas indicate data transmission in the other direction.

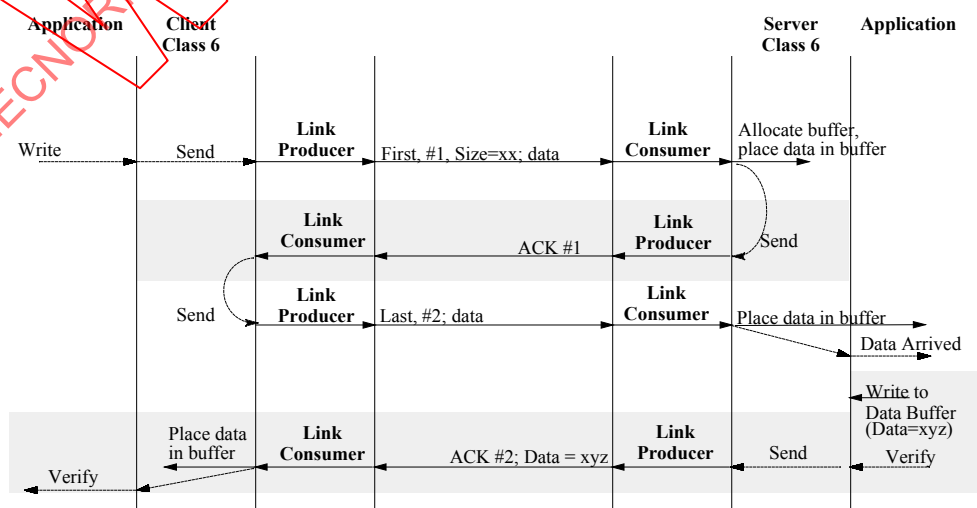


Figure 77 – Sequence diagram of two fragments using transport class 6

Current sequence number (SN);

Try counter;

ACK accumulator (N server nodes).

9.3.12.5.2 States

The defined states and their descriptions of the Class 6 Transport Client are listed in Table 254.

Table 254 – Class 6 transport client states

State	Description
Ready	The instance of Transport Class 6 has been created and started.
Waiting	The instance of Transport Class 6 has been created and started. Application data has been written and the client is waiting for response.

NOTE These two states are sub-states of the Running state described as the common feature for classes 4 to 6.

9.3.12.5.3 State transitions

State transitions of the class 6 client transport is shown in Figure 79.

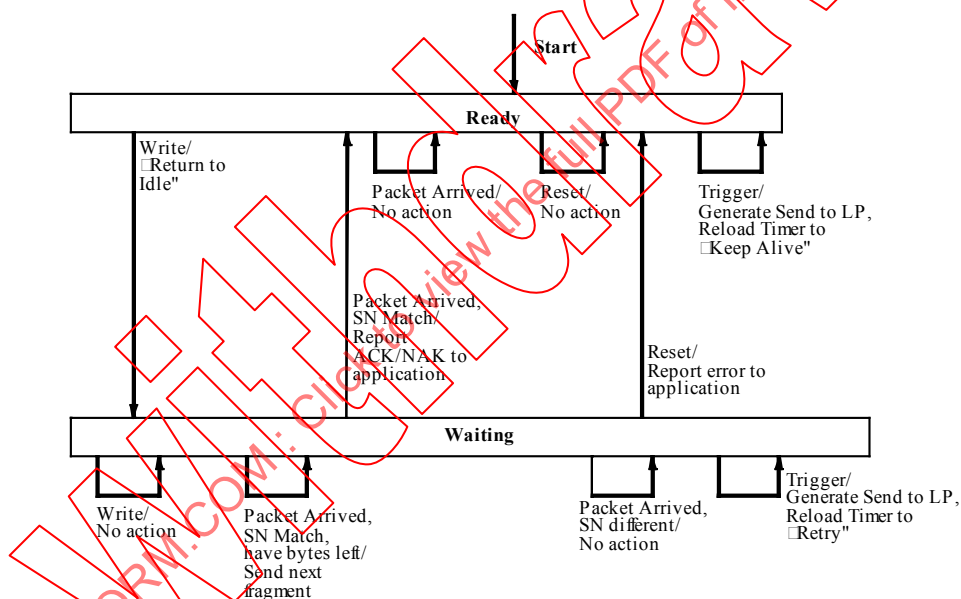


Figure 79 – Class 6 client STD

9.3.12.5.4 State event matrix

State transitions of the class 6 client transport is shown in Table 255.

Table 255 – Class 6 client state event matrix

Event	Command	Sequence number	State	
			Ready	Waiting
Reset	n/a	n/a	No action.	Do “Return to Ready processing” with “Reset” error status.
Write	n/a	n/a	Increment Sequence number. Test if the message fits in one fragment; if it does: Set CMD to <i>Only</i> Copy message to TPDU buffer else: Set CMD to <i>First</i> Calculate (total size / 256 octets) – 1; put into header Copy first fragment to TPDU buffer Set Try counter Load timer with time-out value Generate Send event to Link Producer Go to Waiting state	No action.
NewData	ACK	Different	(Not expecting anything). No action.	(Assume the other transport is ACKing something old) No action.
		Same		If the whole application message has been sent, then do “Return to Ready processing” with Success status. else, if the remainder fits in one fragment, then place the remainder in the Link Producer buffer, increment the sequence count, set the CMD to <i>Last</i>, and generate the Send event to the Link Producer else, place the next fragment in the Link Producer buffer, increment the sequence count, set the CMD to <i>Middle</i>, and generate the Send event to the Link Producer and stay in Waiting state.
	NAK	Different		(Assume the other transport is NAKing something old) No action.
		Same		Do “return to Ready processing” with NAK error status
	Other	Don't care		(Assume other end is confused.) No action.
Trigger (Retry or Keep alive timer)	Don't care	Don't care	Generate Send event to Link Producer (which re-sends the <i>Null</i> message to keep connection alive), reload timer with “keep alive” value.	Decrement Try counter, if zero: Do “return to Ready processing” with “Can't send” error status. else: Generate Send event to Link Producer (which re-sends the previous message), reload timer with retry value.

The “Return to Ready Processing” includes the following:

- Free the application data buffer
- Generate *Verify* event to application with appropriate status
- Increment Sequence number
- Set CMD to *Null*
- Generate the Send event to the Link Producer and reload the timer with the Keep Alive value
- Go to Ready state.

9.3.12.6 Transport class 6 server

9.3.12.6.1 Functions

In the link to application direction, a Class 6 Server transport is responsible for:

- accepting the notification from the consuming node of the network connection that it has written a data packet to the server-side TPDU receive buffer;
- locating and reading the transport headers of each packet that the consuming node of the network connection writes to the server-side TPDU receive buffer;
- performing the actions specified in the state event matrix for each packet which arrives;
- notifying the server application when it places new data into the received data buffer.

The Server Bi-directional Point-to-Point Non-Blocking transport has four states:

Initial;
Ready;
Assembling;
Waiting (for application verification)

The following internal storage is assumed:

Previous Sequence Number (SN);
Previous Command (CMD).

9.3.12.6.2 States

The defined states and their descriptions of the Class 6 Transport Server are listed in Table 256.

Table 256 – Class 6 transport server states

State	Description
Initial	The instance of Transport Class 6 has been created and started.
Assembling	The instance of Transport Class 6 has been created and started. First packet of segmented application data has arrived.
Waiting	The instance of Transport Class 6 has been created and started. Subsequent packet(s) of segmented application data has(ve) arrived.
Ready	The instance of Transport Class 6 has been created and started. Last packet of segmented application data has arrived.

9.3.12.6.3 State transitions

The diagram illustrates the state transitions for the CSMA/CD protocol across four states: Initial, Assembling, Waiting, and Ready.

- Initial State:**
 - Packet Arrived, CMD=Only/Generate "Data Arrived" event:** Transitions to the Assembling state.
 - Packet Arrived, CMD=others/Send NAK:** Transitions to the Initial state.
 - Reset/No action:** Transitions to the Initial state.
 - Packet Arrived, CMD=First/Allocate memory, send ACK:** Transitions to the Assembling state.
 - Reset/Free memory:** Transitions to the Initial state.
 - Packet Arrived, SN same as last time, CMD different/Free memory, send NAK:** Transitions to the Initial state.
- Assembling State:**
 - Packet Arrived, CMD and SN same as last time/Resend last response:** Transitions to the Waiting state.
 - Packet Arrived, CMD=Middle/Put into buffer, send ACK:** Transitions to the Assembling state.
 - Packet Arrived, CMD=First/Free memory, allocate memory, send ACK:** Transitions to the Assembling state.
 - Packet Arrived, CMD=Last/Generate "Data Arrived" event:** Transitions to the Waiting state.
 - Packet Arrived, CMD=Only/Free memory, Generate "Data Arrived" event:** Transitions to the Waiting state.
 - Packet Arrived, not enough room/Free memory, send NAK:** Transitions to the Initial state.
- Waiting State:**
 - Verify/Generate Send event:** Transitions to the Ready state.
 - Packet Arrived, CMD=others/Send NAK:** Transitions to the Waiting state.
 - Packet Arrived, CMD=Only/Generate "Data Arrived" event:** Transitions to the Waiting state.
 - Packet Arrived, CMD=First/Allocate memory, send ACK:** Transitions to the Assembling state.
- Ready State:**
 - Packet Arrived, CMD and SN same as last time/Resend last response:** Transitions to the Waiting state.
 - Packet Arrived, CMD=others/Send NAK:** Transitions to the Ready state.

External transitions include "Reset/Free memory" from the Assembling and Waiting states back to the Initial state.

Figure 80 – Class 6 server STD

The following events can happen.

- *Reset* Event;
- *Verify* Event;
- Packet Arrived Event, CMD= Null;
- Packet Arrived Event, CMD= Only
- Packet Arrived Event, CMD= First;
- Packet Arrived Event, CMD= Middle;
- Packet Arrived Event, CMD= Last;
- *Packet Arrived* Event, CMD= Illegal command.

NOTE There is **no** timer event used by the Server state machine. If there is a communications problem, the Client state machine should time out. The application should have a timer to indicate problems with the other application.

The following internal storage is assumed:

Previous SN;
Previous CMD received;
Buffer Pointer;
Room Left.

The following general guiding principles were used to develop the particular actions and transitions:

- Act like class 3 when possible
- A *Null* shall not change state
- When a *First* or *Only* is received when in the *Assembling* or *Waiting* states, free up the pending message and act like the server was in the *Ready* state
- When a sequence error is detected, free up the resources and go to the *Ready* state (or stay in the *Initial* state)
- It is important to NOT change the CMD when responding to a retry. It might have been NAKed the last time; it shall be NAKed on a retry also.

9.3.12.6.4 State event matrix

State transitions of the class 6 server transport are shown in Table 257.

Table 257 – Class 6 server SEM

Event	State			
	Initial	Ready	Assembling	Waiting
Reset	No action.	Go to Initial.	Free memory previously allocated. Go to Initial.	Free memory previously allocated. Go to Initial.
Verify	Ignore.			Generate Send event, go to Ready.
Packet Arrived, CMD=Null, SN different	Copy SN to From_Recv, set CMD to ACK, generate Send event, go to Ready.	Copy SN to From_Recv, set CMD to ACK, generate Send event, stay in Ready.	Copy SN to From_Recv, set CMD to ACK, generate Send event, stay in Assembling.	Free memory previously allocated. Copy SN to From_Recv, set CMD to NAK (sequence error), generate Send event, go to Ready. (If we ACK the Null, there is no way to ACK the original request. If we don't ACK the Null, the client shall assume we died.)
Packet Arrived, CMD=Null, SN same		If previous CMD was Null, then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Generate Send event, stay in Ready. else: (Assume other transport made an error by sending a different CMD, but used the same SN). Copy SN to From_Recv, set CMD to NAK (sequence error), generate Send event, stay in Ready.	If previous CMD was Null, then: (Assume other transport still has nothing for us to do, assume SN and CMD are OK.) Generate Send event, stay in Assembling. else: (Assume other transport made an error by sending a different CMD, but used the same SN). Free memory previously allocated. Copy SN to From_Recv, set CMD to NAK (sequence error), generate Send event, go to Ready.	
Packet Arrived, CMD=Only, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Data Arrived event, copy SN to From_Recv, set CMD to ACK, go to Waiting. else: Copy SN to From_Recv, set CMD to NAK (no memory), generate Send event, go to Ready.		Free memory previously allocated. Allocate buffer; if buffer allocation succeeds: Copy data to buffer, put on Received queue, generate Data Arrived event, copy SN to From_Recv, set CMD to ACK, go to Waiting. else: Copy SN to From_Recv, set CMD to NAK (no memory), generate Send event, go to Ready.	

Event	State			
	Initial	Ready	Assembling	Waiting
Packet Arrived, CMD=Only, SN same	(Since this state does not know the previous Sequence Number, perform action as if the Sequence Number were different.)	If previous CMD was <i>Only</i>, then: Generate Send event, stay in Ready. (This is a retry, assume SN and CMD are OK. Since we are in Ready, and not Waiting, the application has verified it.) else: Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, stay in Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, go to Ready. NOTE Shall not be in this state if previous CMD was <i>Only</i>.	If previous CMD was <i>Only</i>, then: Generate Duplicate Arrived event, stay in Waiting. (This is a retry, assume SN and CMD are OK.) else: Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, go to Ready.
Packet Arrived, CMD=First, SN different	Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i>, initialize Buffer Pointer and Room Left, set CMD to ACK, generate Send event, go to Assembling state. else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), generate Send event, go to Ready.		Free memory previously allocated. Allocate buffer; if buffer allocation succeeds: Copy data to buffer, copy SN to <i>From_Recv</i>, initialize Buffer Pointer and Room Left, set CMD to ACK, generate Send event, go to Assembling. else: Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), generate Send event, go to Ready.	
Packet Arrived, CMD=First, SN same	(Since this state does not know the previous Sequence Number, perform action as if the Sequence Number were different.)	Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, stay in Ready. NOTE Shall not be in this state if previous CMD was <i>First</i>.	If previous CMD was <i>First</i>, then: Generate Send event, stay in Assembling. (This is a retry, assume SN and CMD are OK.) else: Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, go to Ready. (The Client changed the CMD but not the SN.)	Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, go to Ready. NOTE Shall not be in this state if previous CMD was <i>First</i>.

Event	State			
	Initial	Ready	Assembling	Waiting
Packet Arrived, CMD=Middle, SN different	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Ready.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, copy SN to <i>From_Recv</i>, update Buffer Pointer and Room Left, set CMD to ACK, generate Send event, stay in Assembling. else: Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), generate Send event, go to Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready.
Packet Arrived, CMD=Middle, SN same			If previous CMD was <i>Middle</i>, then: Generate Send event, stay in Assembling. (This is a retry, assume SN and CMD are OK.) else: Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (sequence error), generate Send event, go to Ready.	
Packet Arrived, CMD=Last, SN different	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Ready.	If size of data is less than Room Left, then: Copy data to buffer at Buffer Pointer, put buffer on Received queue, generate Data Arrived event, copy SN to <i>From_Recv</i>, set CMD to ACK, generate Send event, go to Ready. else: Free memory previously allocated. Copy SN to <i>From_Recv</i>, set CMD to NAK (no memory), generate Send event, go to Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready.

Event	State			
	Initial	Ready	Assembling	Waiting
Packet Arrived, CMD=Last, SN same		If previous CMD was <i>Last</i> , then: Generate Send event, stay in Ready. (This is a retry, assume SN and CMD are OK. Since we are in Ready, and not Waiting, the application has verified it.) else: Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, stay in Ready.	Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready. NOTE This shall not happen: If the previous CMD is <i>Last</i> , then we shall be in the Waiting or Ready state.	If previous CMD was <i>Last</i> , then: Generate Duplicate Arrived event, stay in Waiting. (This is a retry, assume SN and CMD are OK.) else: Free memory previously allocated. Copy SN to <i>From_Recv</i> , set CMD to NAK (sequence error), generate Send event, go to Ready.
Packet Arrived, CMD=Illegal	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), generate Send event, stay in Initial.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), generate Send event, stay in Ready.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), generate Send event, stay in Assembling.	Copy SN to <i>From_Recv</i> , set CMD to NAK (illegal command), generate Send event, stay in Waiting.

10 DLL mapping protocol machine 1 (DMPM 1)

10.1 General

The DLL Mapping Protocol Machine is common to all the AREP types.

The network connections that interface to the Data-link layer include the link producer(s) and link consumer(s).

Data flow of the link producer and consumer on an instance of Transport connection is shown in Figure 81.

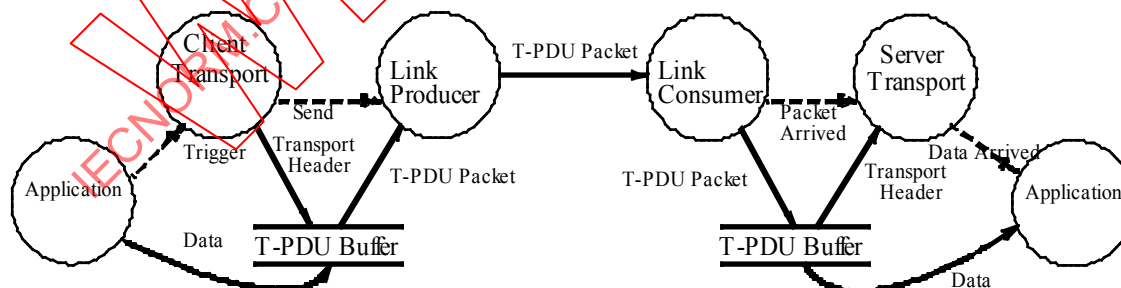


Figure 81 – Data flow diagram for a link producer and consumer

10.2 Link producer

The link producer shall be responsible for fetching data from the TPDU buffer and producing it on the data-link after the *send* event has been received. TPDU buffer management schemes such as overwrite, double buffered, triple buffered are not excluded, but since they are implementation specific, they are not specified here.

The link producer shall read the TPDU packet from the TPDU buffer and transmit it unchanged. While the TPDU buffer is shown as a single buffer, it does not imply that the

transport header and the data are stored in consecutive locations. An implementation that stored the transport header and data in non-consecutive locations is permitted as long as the link producer can find all parts of the TPDU packet.

The application, not the link producer, shall be responsible for ensuring that the TPDU buffer has been initialized before the first trigger has been issued.

10.3 Link consumer

The link consumer is responsible for receiving TPDU packets, storing them in the TPDU buffer and notifying the server transport class that a packet has arrived. Data buffer management schemes such as overwrite, double buffered, triple buffered are not excluded, but since they are implementation specific, they are not specified here.

The link consumer shall write the TPDU packet unchanged to the TPDU buffer. While the TPDU buffer is shown as a single buffer, it does not imply that the transport header and the data are stored in consecutive locations. There are no restrictions on actual implementations. An implementation that stored the transport header and data in non-consecutive locations is permitted as long as the consumer can find both parts of the TPDU packet.

10.4 Primitive definitions

10.4.1 Primitives exchanged between DMPM and ARPM

List of primitives exchanged between DMPM and ARPM is shown in Table 258 and Table 259.

Table 258 – Primitives issued by ARPM to DMPM

Primitive names	Source	Associated parameters	Functions
Send.req	ARPM	Service Dest-id TPDU	This primitive is used to request the DMPM to transfer the TPDU to the Data-link. It passes the TPDU to the DMPM as a DLSDU.

Table 259 – Primitives issued by DMPM to ARPM

Primitive names	Source	Associated parameters	Functions
Packet_Arrived.ind	DMPM	Service Source-id TPDU	This primitive is used to pass the TPDU received as a data-link layer service data unit to a designated ARPM.

10.4.2 Parameters of ARPM/DMPM primitives

The parameters used with the primitives exchanged between the ARPM and the DMPM are described in Table 260.

Table 260 – Parameters used with primitives exchanged between ARPM and DMPM

Parameter name	Description
TPDU	This parameter carries the transport PDU containing the user data and Transport header.
DL-SDU	This parameter carries the received Data-link SDU.

10.4.3 Primitives exchanged between data-link layer and DMPM

The request and confirmation services used to queue Lpackets with the data-link layer shall be of the form:

NOTE The primitives shown in Table 261 and their parameters are defined in detail in Type 2 clauses of IEC 61158-3-2:2007.

Table 261 – Primitives exchanged between data-link layer and DMPM

Primitive names	Source	Associated parameters
DL_generic_Tag.req	ARPM	Request_DLS_user_id (handle) user_data [] QoS (priority) generic_TAG
DL_generic_Tag.cnf	Data-link layer	Request_DLS_user_id (handle) TX_status
DL_generic_Tag.ind	Data-link layer	Request_DLS_user_data [] generic_TAG
DL_fixed_Tag.req	ARPM	Request_DLS_user_id (handle) user_data [] QoS (priority) fixed_TAG destination-DLE-ID
DL_fixed_Tag.cnf	Data-link layer	Request_DLS_user_id (handle) TX_status
DL_fixed_Tag.ind	Data-link layer	user_data [] fixed_TAG source-DLE-ID
DL_tone.ind	Data-link layer	DLS_cycle

10.4.4 Parameters of DMPM/Data-link Layer primitives

The parameters used with the primitives exchanged between the DMPM and the data-link layer are identical to those defined in the Type 2 clauses of data-link layer Service Definition (IEC 61158-3-2:2007).

The parameters are described in Table 262.

Table 262 – Parameters used with primitives exchanged between DMPM and Data-link

Parameter name	Description
Request_DLS_user_id	Provides a local means of pairing the resulting confirm primitive with the causative request primitive.
User_data	Data to be transmitted between DLS-users without alteration by the DLS-provider.
QoS	Selects one of the following priorities : scheduled high low
Generic_TAG	Connection identification or service point address identifying the remote DLSAP(s) to which the DLS is to be provided.
Fixed_TAG	Specifies the destination service point in the station identified by the DLS- Destination-DLE-ID address.
Destination-DLE-ID	Address identifying the destination station for the message using its MAC ID address.
Source-DLE-ID	Address identifying the local station from which the fixed tag DLSDU has been sent. It is a station MAC ID address on the local link.
TX_status	Status of the requested transmission : a) "OK" message successfully sent; b) "TxAbort" sending process failed; c) "Flushed" message has been removed from the pending queue before sending.
DLS_cycle	Interval count for the Network Update Time (NUT) which has just been received within the overall cycle of scheduled access intervals.

10.5 DMPM state machine

10.5.1 DMPM states

10.5.1.1 Link producer

10.5.1.1.1 Link producer states

The defined states and their descriptions of the Link Producer are listed Table 263 below :

Table 263 – Link producer states

State	Description
Non-existent	The Link Producer does not exist.
Running	The Link Producer has been created and is running.

10.5.1.1.2 State transition diagram

Figure 82 shows the state transition diagram of a link producer.

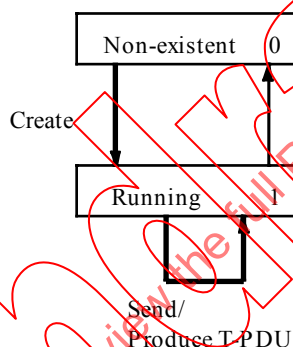


Figure 82 – State transition diagram for a link producer

10.5.1.1.3 State event matrix

State event matrix of the Link Producer is shown in Table 264.

Table 264 – State event matrix of link producer

Event	State	
	Non-existent	Running
Create	Transition to Running	Error
Delete	Error	Transition to Non-existent
Send	Error	1) Produce TPDU

10.5.1.2 Link consumer

10.5.1.2.1 Link consumer states

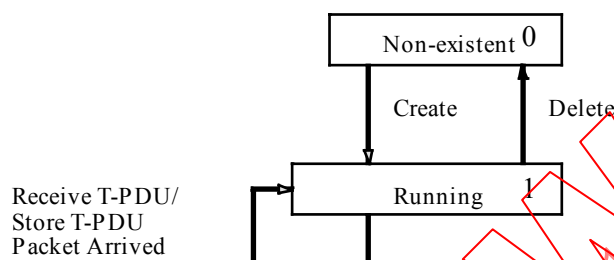
The defined states and their descriptions of the Link Consumer are listed in Table 265 below :

Table 265 – Link consumer states

State	Description
Non-existent	The Link Producer does not exist.
Running	The Link Producer has been created and is running.

10.5.1.2.2 State transition diagram

Figure 83 shows the state transition diagram of a link consumer.

**Figure 83 – State transition diagram for a link consumer****10.5.1.2.3 State event matrix**

State event matrix of the Link Producer is shown in Table 266.

Table 266 – State event matrix of link consumer

Event	State	
	Non-existent	Running
Create	Transition to Running	Error
Delete	Error	Transition to Non-existent
Receive TPDU	Error	Store TPDU Packet Arrived

10.5.2 Functions used by DMPM

This fieldbus does not any specified functions.

10.6 Data-link Layer service selection

This fieldbus supports only the Data-link layer service whose primitives are noted in 10.4.3.

11 DLL mapping protocol machine 2 (DMPM 2)**11.1 General**

This subclause defines the requirements for mapping the transmission of TPDU's over an Ethernet-TCP/UDP/IP-based network, instead of the Type 2 data-link layer, using the TCP/UDP/IP protocol suite and the encapsulation protocol defined in 4.3.1.

11.2 Mapping of UCMM PDUs

11.2.1 General

UCMM PDUs shall be transmitted over a TCP/IP connection, using the encapsulation protocol defined in 11.8 and 4.3.1 (SendRRData command).

11.2.2 Common requirements for Connection Manager PDU's

11.2.2.1 General

This subclause contains the specific requirements for Connection Manager parameters in case of TCP/IP encapsulation (Connection Manager parameters are fully described in 4.1.5.2).

11.2.2.2 Connection type

The connection type shall be NULL, MULTICAST, or POINT2POINT. The MULTICAST connection type shall be supported only for transport class 0 and class 1 connections.

11.2.2.3 Priority

The priority shall be LOW, or HIGH, or SCHEDULED. At present, SCHEDULED priority shall be treated no differently than HIGH priority.

Targets and originators shall close any transport class 2 or 3 connections when the corresponding originating TCP connection is closed.

NOTE SCHEDULED priority for CP 2/2 connections may be further defined in the future.

11.2.2.4 Trigger type

The trigger type shall be CYCLIC, CHANGE_OF_STATE, or APPLICATION. Transport class 0 and class 1 connections that use CHANGE_OF_STATE triggering shall use the production inhibit time segment (see 4.1.9.5).

11.2.2.5 Connection size

The connection size shall be no larger than 65 511 octets.

NOTE The Forward_Open request limits the connection size to 511 octets; however, the optional Large_Forward_Open allows larger connection sizes.

11.2.2.6 Connection request timeout

To reliably establish a connection that extends onto a TCP/IP link, the connection request time-out shall be large enough to allow the connection to be established, which could involve resolving a host name, or going through multiple gateways.

Because of the large variation in connection request processing over TCP/IP, Type 2 routers in the connection path shall not subtract anything from the connection request timeout.

11.2.2.7 Connection path

The link address portion of a TCP/IP connection path segment shall be encoded within a port segment as specified in 4.1.9.3.

11.2.2.8 Network Connection ID

For TCP/IP connections, the Network Connection ID shall be a 32-bit identifier meaningful to the device that chooses it. The Network Connection ID need not be subdivided into any specific fields.

In general, the consuming device selects the Network Connection ID for a point-to-point connection, and the producing device selects the Network Connection ID for a multipoint connection. Table 267 shows which device, Target or Originator, shall choose the T⇒O and O⇒T Network Connection IDs.

Table 267 – Network Connection ID selection

Connection type	Which network Connection ID	Who chooses Connection ID
Point-to-point	Originator -> Target	Target
	Target -> Originator	Originator
Multicast	Originator -> Target	Originator
	Target -> Originator	Target

The Network Connection ID shall not be reused until the connection has been closed or has timed out. When a device restarts, it shall not reuse Network Connection IDs from previously opened connections until those connections have been closed or have timed out. A specific connection ID shall not be reused so long as there is the possibility that packets with that connection ID are present in the network.

11.2.3 Forward_open PDU for class 2 and class 3 connections

The Forward_open PDU for class 2 and class 3 connections shall be sent over a TCP connection using the SendRRData command defined in 4.3.1.

11.2.4 Forward_open for class 0 and class 1 connections

11.2.4.1 General

The Forward_Open PDU for transport class 0 and class 1 connections shall be sent over a TCP connection using the SendRRData command defined in 4.3.1. As part of the Forward_Open dialog, the producer and consumer shall exchange the UDP port numbers and IP multicast address (for multipoint connections) necessary to send the transport class 0 and class 1 connected data. The Sockaddr Info item defined in 4.3.1 shall be used to encode the UDP port numbers and IP multicast address. The use of the Sockaddr Info item varies depending on whether the connection is multipoint or point-to-point, and whether the connection originator or the connection target is the producer.

For multipoint connections, the producer shall choose an IP multicast address to which to send the connected data. The port number shall be the registered UDP port number (0x08AE) assigned by the IANA. The IP multicast address and UDP port number shall be encoded via a Sockaddr Info item. A Sockaddr Info item shall be sent with Forward_Open (if the connection originator is the producer), or with the Forward_Open response (if the connection target is the producer).

For point-to-point connections, the consumer shall choose a UDP port number to which the connected data shall be sent. The port number may be the registered port number (0x08AE) or may be a port number chosen by the consumer. The port number shall be encoded in a Sockaddr Info item and shall be sent with the Forward_Open (if the connection originator is the consumer), or with the Forward_Open response (if the connection target is the consumer).

The Sockaddr Info item(s) shall be placed after the Forward_Open and/or Forward_Open response data in the SendRRData command/reply. If the Sockaddr Info items are not present, or are in error, a Forward_Open response shall be returned with status code 0x01 and extended status 0x205.

NOTE After receiving the Forward_open response for a multipoint connection, the consuming Ethernet devices should join the desired IP Multicast Group in order to receive the IP Multicast datagrams. The exact method for doing this depends on the TCP/IP application programming interface in use on the device.

11.2.4.2 Mapping connections to IP multicast addresses

NOTE It is recommended, though not required, that producers use a unique IP multicast address for each active multipoint connection. Depending upon the implementation, this can reduce the amount of connection screening on the part of the consumer. It also allows the consumer to more evenly service incoming connected data from multiple connections.

Since a unique IP multicast addresses per multipoint connection is not required, consumers shall be able to handle the situation in which packets from multiple multipoint connections are being sent to the same IP multicast address. Consumers shall be able to screen the incoming packets based on the Connection ID and source IP address (see 11.3.4 for screening requirements).

11.2.4.3 IP multicast scoping and address allocation

11.2.4.3.1 General

The following subclauses addresses two issues that shall be considered when implementing multipoint connections on Ethernet:

- IP multicast scoping refers to the practice of limiting how widely a given multicast datagram is propagated across the network;
- IP multicast address allocation refers to the problem of how applications select IP multicast addresses that are used to send and receive IP multicast datagrams.

NOTE Once the IP multicast allocation architecture becomes standard and is deployed, requirements detailed in the two following subclauses may be updated as appropriate.

11.2.4.3.2 IP multicast scoping practices (informative)

In general, most currently deployed networks use the practice of “TTL scoping” in conjunction with router and/or switch configuration to confine multicast traffic to desired network boundaries.

TTL scoping refers to the practice of using the “Time to Live” (TTL) field in the IP header to limit the number of network hops over which the multicast packet is propagated. When sending an IP multicast datagram, a host can set the TTL field in the IP header to an appropriate value based on how widely the datagram should be propagated. As the datagram is routed through the network, each hop decrements the TTL field. Routers can be configured with TTL thresholds such that they will not forward a packet unless the TTL is greater than the threshold.

A multicast datagram with an initial TTL of 1 limits the datagram to the local subnet. Other common TTL values are 16 for multicast within a site and 64 for multicast within a region.

In addition to TTL scoping, multicast routing protocols and other methods are commonly used to control the propagation of multicast traffic. Routers commonly support multicast protocols such as PIM, DVMRP. Switches that implement “IGMP snooping” can limit the multicast packets sent on a port to only those multicast addresses for which the end device has issued an IGMP membership message. Configuration of switches and routers is usually done by knowledgeable staff.

11.2.4.3.3 IP multicast address allocation practices (informative)

The entire IP multicast address space is 224.0.0.0 to 239.255.255.255. The Internet Assigned Numbers Authority (www.iana.org) is responsible for allocation of the IP multicast address space. IP multicast addresses have been assigned to particular organizations, and for particular protocols. In addition there is a large block of IP multicast addresses allocated for “administratively scoped” multicast, from which applications may allocate addresses, and for which a suite of allocation and scoping protocols are being developed by the Internet Engineering Task Force (IETF). The administratively scoped range is from 239.0.0.0 through 239.255.255.255 (and is further partitioned into additional ranges).

Unfortunately at present there are no widely deployed standard mechanisms for allocating and assigning multicast addresses to applications. For example, when a network administrator deploys a video streaming application, the application will have its own specific mechanism for assigning IP multicast addresses.

11.2.4.3.4 Multicast scoping for Type 2

By default, Type 2 devices shall use a TTL equal to 1 for transport class 0 and 1 multicast packets. The use of a TTL value of 1 prevents multicast packets from propagating beyond the local subnet. When TTL is equal to 1, both the Type 2 producer and consumer shall be on the same subnet.

Type 2 devices are strongly encouraged to support the explicit configuration of the TTL value for IP multicast packets. If supported, devices shall use the TCP/IP Interface object (class 0xF5) as the mechanism to configure the TTL value. When a TTL value greater than 1 is configured, then the producer and consumer may be on different subnets.

If the TTL value has not been configured to be greater than 1 and if a multipoint connection request is received from an originator on a different subnet, then the device shall return general status 0x01 and extended status 0x813 in the Forward_Open response.

When the TTL value is explicitly configured, it shall be used for all Type 2 multicast packets.

11.2.4.3.5 Multicast address allocation for Type 2

11.2.4.3.5.1 General

Two mechanisms are defined for allocation of IP multicast addresses used for Type 2 multicast packets: using an algorithm based on the device's IP address, and explicit configuration via the TCP/IP Interface object. Type 2 devices shall implement the algorithm-based method by default. Devices are also strongly encouraged to implement the method for explicit configuration of multicast addresses. Both methods are described below.

11.2.4.3.5.2 Allocation algorithm based on the device's IP address

The overall IP multicast address range shall be the Organizational Local Scope, and shall start at 239.192.1.0. Each device shall use a block of (at most) 32 multicast addresses from this range. Each device shall calculate the block of multicast addresses via an algorithm, described further below, that uses the Host Id portion of the device's IP address.

A device's Host Id shall be determined by applying the subnet mask to the device's IP address. If a subnet mask is configured for the device, the subnet mask shall be applied to the IP address to determine the Host Id. If no subnet mask is in use, then the class of the device's IP address shall be used to determine the Host Id (e.g., for Class C addresses 8 bits of Host Id shall be used, for Class B addresses 16 bits of host id shall be used, and for Class A addresses 24 bits of Host Id shall be used).

In order to keep the IP multicast addresses within the IPv4 Organization Local Scope, and to put a reasonable bounds on the number of multicast addresses in use, devices shall use at most the low-order 10 bits of the Host Id in generating the range of multicast addresses. This allows for 1 024 unique Host Ids.

The following pseudo-code shows the algorithm to determine the device's starting and ending multicast addresses:

```
Type2_Mcast_Base_Addr = 0xEFC00100 // 239.192.1.0 is the starting address
Type2_Host_Mask = 0x3FF // 10 bits of host id

if Subnet_mask configured then
```

```

Netmask = Subnet_mask
else
  if IP_address is Class A then
    Netmask = 255.0.0.0
  else if IP_address is Class B then
    Netmask = 255.255.0.0
  else if IP_address is Class C then
    Netmask = 255.255.255.0
end_else

Host_id = IP_addr & (~Netmask)
Mcast_index = Host_id - 1
Mcast_index = Mcast_index & (Type2_Host_Mask)

Mcast_start_addr = Type2_Mcast_Base_Addr + (Mcast_index * 32)
Mcast_end_addr = Mcast_start_addr + 31

```

Table 268 shows example multicast assignments.

Table 268 – Example multicast assignments

Subnet mask	IP address	Multicast addresses
None configured (8 bits of Host Id used, since 192.168.x.x is Class C)	192.168.1.1	239.192.1.0 - 239.192.1.31
	192.168.1.2	239.192.1.32 - 239.192.1.63
	192.168.1.3	239.192.1.64 - 239.192.1.95
255.255.248.0 (11 bits of Host Id)	10.10.16.1	239.192.1.0 - 239.192.1.31
	10.10.16.2	239.192.1.32 - 239.192.1.63
	10.10.16.3	239.192.1.64 - 239.192.1.95
	10.10.20.0 (Host Id = 1 024; lower 10 bits all 0)	239.192.128.224 - 239.192.128.255 (this is the highest multicast address range that would result using the algorithm)

Since there are a finite number of unique Host Ids, it is possible for different devices to produce data using the same multicast address. Consequently, devices that receive packets on Type 2 multicast connections shall screen the incoming packets based on the Connection Id as well as the IP address of the sending device. This is described in 11.3.4.

When the device's IP address or subnet mask changes, the IP multicast addresses generated by the algorithm also shall change accordingly.

11.2.4.3.5.3 Explicit configuration of IP multicast addresses

IP multicast addresses are configured via the TCP/IP Interface object (class 0xF5). The user (or software) can configure a starting multicast address and number of multicast addresses to allocate. The configured multicast addresses shall then be used for Type 2 multicast packets.

Type 2 devices shall at least implement the algorithmic method for allocating IP multicast addresses, and are encouraged to implement both methods. If both methods are implemented, the algorithmic method shall be the default "out of box" method.

11.3 Mapping of transport class 0 and class 1 PDUs

11.3.1 UDP datagrams

PDUs for transport class 0 and class 1 connections shall be encapsulated in UDP datagrams using the Common Packet Format defined in 4.3.3. PDUs for multipoint connections shall be transmitted using IP multicast.

The data portion of the UDP datagram for class 0 and class 1 PDUs shall be as shown in Table 269.

Table 269 – UDP data format for class 0 and class 1

Field name	Type	Value
Item Count	UINT	2
Type ID	UINT	0x8002 (Sequenced Address Type)
Length	UINT	8
Address Data	UDINT	Connection ID (from Forward_open response)
	UDINT	Sequence Number
Type ID	UINT	0x00B1 (Connected Data Type)
Length	UINT	Number of octets in PDU to follow
Data		class 0 or class 1 PDU

11.3.2 No linkage with TCP connections

In order to open a transport class 0 or 1 connection, a TCP connection and an encapsulation session shall first be established. The TCP connection is used to send the Forward_Open request and receive the Forward_Open response. Once the TCP connection is opened, and transport class 0 and 1 connections are established, it is recommended that Type 2 devices leave the TCP connection open. If the TCP connection is left open, it is then available for subsequent communications such as a Forward_Close or other explicit messages.

Although it is recommended that devices leave the TCP connection open, there shall be no linkage between the TCP connection used to open a transport class 0 or class 1 connection and the resulting class 0 or class 1 connection. If a TCP connection closes, the closing of the TCP connection shall not cause the target or the originator to close any corresponding transport class 0 or class 1 connections.

11.3.3 Class 0 and class 1 packet ordering

NOTE 1 By definition class 0 and class 1 transports do not detect out-of-order packets. For class 0, every packet is considered to be new data. For class 1, only duplicate data is detected. If a new packet arrives and the sequence number in the transport PDU is different from the previous packet, then the new packet is considered to be new data even if the new packet has a sequence number that is less than the previous sequence number.

NOTE 2 When using UDP to transport Class 0 and class 1 connected data, there is no guarantee that packets arrive in the same order that they were sent. When both sender and receiver are on the same subnet, packets typically arrive in order. However, when going through routers, when there are multiple paths that a packet could take, it is possible for packets to arrive out of order.

For class 0 and class 1 connections over Ethernet, devices shall maintain a sequence number in the data portion of the UDP datagram defined in 11.3.1. The sequence number shall be maintained per connection. Each time an Ethernet device sends a Class 1 packet, it shall increment the sequence number for that connection. If the receiving Ethernet device receives a packet whose sequence number is less than the previously received packet, the packet with the smaller sequence number shall be discarded. Duplicate packets shall be accepted and given to the transport layer.

The sequence number shall be operated on with modular arithmetic to deal with sequence rollover.

NOTE 3 Dealing with 32-bit sequence numbers is described in IETF RFC 793 (the TCP definition).

11.3.4 Screening incoming connected data

Ethernet devices that receive class 0 and class 1 connected data shall screen incoming packets based on the Connection ID and IP address of the sending device. This is necessary for the following reasons.

- For multipoint connections, there is no guaranteed mechanism to prevent multiple devices from using the same IP multicast address. Consequently, a device could receive (bogus) multicast connected data from a device with which it has not established a connection.
- For multicast connections, a device is allowed to use the same IP multicast address for multiple class 0 and class 1 multipoint connections.
- To prevent Connection ID conflicts.

When a class 0 or class 1 connection is established, the target and originating Ethernet devices shall record the Connection ID on which they will receive connected data, coupled with the IP address of the device at the other end of the connection. When a device receives connected data, it shall confirm that the Connection ID is valid for the IP address of the sending device. If not, the packet shall be discarded.

11.4 Mapping of transport class 2 and class 3 PDU's

PDUs for transport class 2 and class 3 connections shall be transmitted over a TCP connection using the encapsulation protocol defined in 11.8 and 4.3.1 (SendUnitData command).

Multiple connections may be sent over a single TCP connection. An implementation need not support a specific number of connections per TCP connection. An implementation may impose an upper bound if it chooses.

Because of the full-duplex nature of TCP, the O2T and T2O link connections shall use the same TCP connection. However if a target subsequently originates a connection, then it shall be considered an originator, and a different TCP connection shall be used.

NOTE This standard defines no requirements for management of the TCP connection, such as inactivity timeouts, or closing the TCP connection when all native connections are closed. However, implementations are free to implement these.

11.5 Mapping of transport classes 4 to 6

The encapsulation protocol defined in 11.8 and 4.3.1 shall not be used to encapsulate PDUs for transport classes 4, 5 and 6.

11.6 IGMP Usage

11.6.1 Background (informative)

The Internet Group Management Protocol (IGMP) is a standard protocol used by hosts to report their IP multicast group memberships and must be implemented by any host that wishes to receive IP multicast datagrams. IGMP messages are used by multicast routers to learn which multicast groups have members on their attached networks. IGMP messages are also used by switches capable of supporting "IGMP snooping" whereby the switch listens to IGMP messages and only sends the multicast packets to ports that have joined the multicast group.

There are two versions of IGMP:

- IGMP V1 is defined in IETF RFC 1112. It defines two messages: Host Membership Query and Host Membership Report (commonly referred to as a “join”)
- IGMP V2 is defined in IETF RFC 2236. It defines additional messages and behavior, notably the Leave Group message.

IGMP V2 is backward compatible with V1. IETF RFC 2236 discusses the interaction between IGMP V1 and V2 hosts and routers.

Since Type 2 devices make extensive use of IP multicast for transport class 0 and 1 connections, consistent IGMP usage by Type 2 devices is essential in order to create well-functioning Type 2 application networks.

11.6.2 IGMP Membership Report messages

Type 2 devices shall issue a Membership Report message when opening a Type 2 connection on which they will receive multicast packets. Specifically, devices shall adhere to the following behavior.

1. When the T->O Connection Type is multipoint (originator is multipoint consumer), the originator shall issue a Membership Report upon receipt of a successful Forward_Open response. The Membership Report shall include the IP multicast address as communicated in the Forward_Open response.
2. When the O->T Connection Type is multipoint (target is multipoint consumer), the target shall issue a Membership Report upon sending a successful Forward_Open response. The Membership Report shall include the IP multicast address as communicated in the Forward_Open.

If the device has already issued a Membership Report for the IP multicast address (e.g. if the multicast address is being used with an existing connection), the device may, but is not required to, issue another Membership Report.

Devices shall also send Membership Report messages in response to Membership Query messages, per the IGMP RFCs.

11.6.3 IGMP Leave Group messages

Devices that support IGMP V2 shall issue a Leave Group when all the Type 2 connections associated with a consuming IP multicast address have either closed or timed out. Specifically, devices shall adhere to the following behavior:

- When the T->O Connection Type is multipoint (originator is multipoint consumer), the originator shall issue a Leave Group upon receipt of a successful Forward_Close response if the originator has no other open connections consuming on that IP multicast address.
- When the O->T Connection Type is multipoint (target is multipoint consumer), the target shall issue a Leave Group upon sending a successful Forward_Close response if the target has no other open connections consuming on that IP multicast address.
- In the event of a connection timeout, the multipoint consumer (whether target or originator) shall issue a Leave Group message if the multipoint consumer has no other connections consuming on that IP multicast address.

11.7 Quality of Service (QoS) for CP 2/2 messages

11.7.1 Overview

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

Within the CPF 2 and CP 2/2 application context, QoS is especially important for time sensitive applications where packet delivery will affect application stability.

CP 2/2 defines requirements and recommendations for how CP 2/2 devices use two standard QoS mechanisms: IEEE 802.1D/IEEE 802.1Q and Differentiated Services. The following summarizes the QoS mechanisms defined for CP 2/2 devices.

- The overall approach is for CP 2/2 end devices to mark CP 2/2 packets with priority values, using the standard mechanisms mentioned above. Marking packets with priority enables infrastructure devices (e.g., switches and routers) to better differentiate CP 2/2 traffic.
- For CPF 2 transport class 0 and 1 connections (i.e., UDP-based), there is a defined mapping of CPF 2 priorities to IEEE 802.1D priorities and DiffServ Code Points (see 11.7.4).
- For UCMM and CPF 2 transport class 3 connections (i.e., TCP-based), there is a defined DiffServ Code Point, and a defined IEEE 802.1D priority value.
- For PTP (IEC 61588) messages, there are DiffServ Code Points and IEEE 802.1D priority values corresponding to the two different types of PTP messages.
- It is strongly recommended that devices by default mark CPF 2 and PTP messages with DSCP values.
- In addition to DSCP, devices may optionally support sending and receiving IEEE 802.1Q tagged frames. If supported, sending tagged frames shall be disabled by default in order to prevent device interoperability problems. When IEEE 802.1Q tagging is desired, the end user shall explicitly enable the sending of IEEE 802.1Q frames, and shall ensure that both the sending and receiving devices support tagged frames.
- The QoS Object provides a means to configure DSCP values, and a means to enable/disable sending of IEEE 802.1Q tagged frames (see IEC 61158-4-2:2010, 7.10).
- There are no requirements for devices to mark traffic other than CPF 2 or IEC 61588. Devices may do so, depending on their implementation capabilities.
- At present there are no specific implementation requirements for end devices to internally differentiate traffic with different priorities. Traffic differentiation in end devices is an area of future development. CP 2/2 implementations are at a minimum recommended to give higher priority to processing CP 2/2 implicit messages over explicit messages. Further differentiation based on the above QoS mechanisms, where possible, is also recommended.

NOTE The following references are applicable to QoS for CP 2/2 devices: IEEE 802.1D, IEEE 802.1Q, IETF RFC 2474, IETF RFC 2475, IETF RFC 2597, IETF RFC 3140, IETF RFC 3246, IETF RFC 4594.

11.7.2 DSCP format

Differentiated Services (DiffServ) is a model for specifying the relative priority of traffic based on the type of service (ToS) field of an IPv4 packet. The model is defined in IETF RFC 2475. DiffServ allows nodes to route packets based on class of traffic as defined by the DiffServ Codepoint (DSCP) and the defined Per-Hop Behavior (PHB) characteristics.

Figure 84 shows the DS field in the IP header.

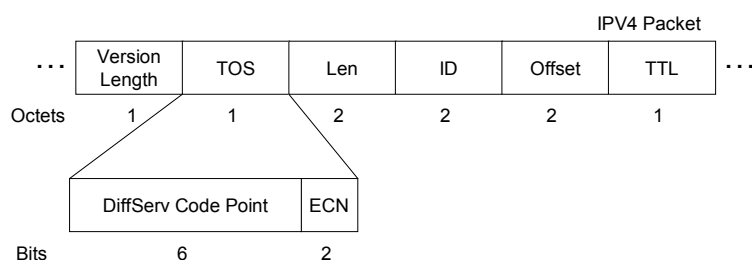


Figure 84 – DS field in the IP header

When setting the DSCP value in the IP header, if placed directly in the ToS field, it shall be shifted left 2 bits.

11.7.3 IEEE 802.1Q/Q format

IEEE 802.1Q defines an Ethernet frame format that allows inclusion of VLAN ID and priority. The IEEE 802.1Q frame has EtherType of 0x8100 and a 4-octet prefix between the Source and Type fields of the frame. The tagged frame defines a 3-bit field to specify 8 priority levels, further specified in IEEE 802.1D. Priority 7 is the highest. Priority 0 is the lowest.

Figure 85 shows the format of an IEEE 802.1Q frame.

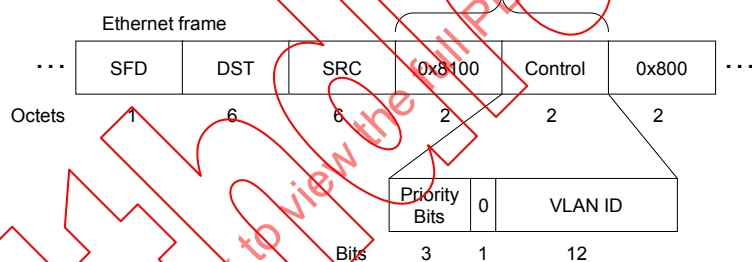


Figure 85 – IEEE 802.1Q tagged frame

11.7.4 Mapping CPF 2 traffic to DSCP and IEEE 802.1D

Table 270 defines the default DSCP and IEEE 802.1D priority mappings for CP 2/2 and CP 2/2.1 (IEC 61588) traffic.

Table 270 – Default DSCP and IEEE 802.1D mapping

Traffic type	CPF 2 priority	DSCP	IEEE 802.1D Priority ^a	CPF 2 traffic usage (recommended)
PTP Event (IEC 61588)	n/a	59 ('111011')	7	n/a
PTP General (IEC 61588)	n/a	47 ('101111')	5	n/a
CPF 2 class 0/1	Urgent (3)	55 ('110111')	6	CIP Motion
	Scheduled (2)	47 ('101111')	5	Safety I/O, I/O
	High (1)	43 ('101011')	5	I/O
	Low (0)	31 ('011111')	3	No recommendation at present
CPF 2 UCMM CPF 2 class 3 All other CP 2/2 encapsulation messages	All	27 ('011011')	3	CPF 2 messaging

^a Sending IEEE 802.1Q tagged frames is disabled by default.

NOTE 1 In Table 270, the IEEE 802.1D and DSCP values for transport class 0 and class 1 messages are based on the CPF 2 priority (indicated in the Forward Open service). PTP and CPF 2 explicit messages use IEEE 802.1D and DSCP values independent of CPF 2 priority.

The default DSCP values and IEEE 802.1Q tagged frame enable/disable may be changed via the QoS Object (see IEC 61158-4-2:2010, 7.10).

NOTE 2 The default DSCP values are "local use" values, as defined in IETF RFC 2474.

11.7.5 CP 2/2 usage of DSCP

When marking CP 2/2 traffic with DSCP values, CP 2/2 devices shall use the values as configured in the QoS Object (default values are shown in Table 270). Usage of the ECN field in the IP header by CP 2/2 end devices is not currently defined and shall be set to 0.

Devices are strongly encouraged to support marking CP 2/2 packets with DSCP values. When supported, the default behavior shall be to mark packets.

All CP 2/2 devices shall support receiving packets with non-zero DSCP values since this is an Internet Protocol requirement (see IETF RFC 791 and IETF RFC 1122). Ethernet infrastructure devices (e.g., switches and routers) can potentially alter DSCP values. Receiving devices shall not assume or otherwise check that incoming DSCP values are the same as in Table 270, or are the same values as sent from the sending device.

11.7.6 CP 2/2 usage of IEEE 802.1D/Q

When sending traffic with IEEE 802.1Q tagged frames, CP 2/2 devices shall use the priority values specified in Table 270. The VLAN ID shall be set to 0, unless a specific VLAN ID for the device has been configured by means not covered by this specification. When receiving IEEE 802.1Q tagged frames, devices shall not require that the VLAN ID be set to 0.

When sending IEEE 802.1Q tagged frames, devices shall also set the corresponding DSCP value in the IP header.

When IEEE 802.1Q frames are supported, the device shall also support the QoS Object (see IEC 61158-4-2:2010, 7.10). The default behavior shall be to disable sending of IEEE 802.1Q tagged frames, since sending tagged frames by default can result in device interoperability problems.

11.7.7 User considerations with IEEE 802.1D/Q

Some CP 2/2 devices do not support receiving IEEE 802.1Q tagged frames. When IEEE 802.1Q frames are sent to such devices, the frames will be dropped. In addition, some managed switches will drop IEEE 802.1Q tagged frames unless the receiving port is configured to accept them.

In order to prevent interoperability problems, sending IEEE 802.1Q frames is disabled by default for CP 2/2 devices. The end user may elect to enable sending tagged frames when supported, via the QoS Object. The user is ultimately responsible for ensuring that both the sending and receiving devices support IEEE 802.1Q frames, and that network infrastructure is properly configured.

11.8 Management of an encapsulation session

11.8.1 Phases of an encapsulation session

An encapsulation session shall have three phases:

- establishing a session;
- maintaining a session;

- closing a session.

11.8.2 Establishing a session

Session establishment shall proceed according to the following steps:

- the originator shall open a TCP/IP connection to the target, using the reserved port number (0xAF12), or if specified, the port number from the connection path;
- the originator shall send a RegisterSession command to the target;
- the target shall check the protocol version in the command message to verify it supports the same protocol version as the originator. If not, the target shall return a RegisterSession with an appropriate Status field along with the highest supported protocol version (see description of RegisterSession command).
- the target shall assign a new (unique) Session ID and shall send a RegisterSession reply to the originator

NOTE A TCP port number, 0xAF12, has been reserved with the Internet Assigned Numbers Authority (IANA) for use by the encapsulation protocol.

11.8.3 Terminating a session

Either the originator or the target may terminate the session. Sessions shall be terminated in either of two ways:

- the originator or target shall close the underlying TCP connection. The corresponding target or originator shall detect the loss of the TCP connection, and shall close its side of the connection;
- the originator or target shall send an UnRegisterSession command and shall wait to detect the closing of the TCP connection. The corresponding target or originator shall then close its side of the TCP connection. The sender of the UnRegisterSession shall detect the loss of the TCP connection, then it shall close its side of the connection.

NOTE The second method is preferred since it results in more timely clean up of the TCP connection.

11.8.4 Maintaining a session

Once a session is established, it shall remain established until one of the following occurs:

- the originator or target closes the TCP connection;
- the originator or target issues the UnRegisterSession command;
- the TCP connection is broken.

12 DLL mapping protocol machine 3 (DMPM 3)

This DMPM is specified in IEC 62026-3:2008, Clause 5.

Bibliography

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

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

IEC/TR 61158-1:2010³, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

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

ISO/IEC 9506-2, *Industrial automation systems – Manufacturing Message Specification – Part 2: Protocol specification*

ISO/IEC 8649, *Information technology – Open Systems Interconnection – Service definition for the Association Control Service Element*

ISO 8650 (all parts), *Information technology – Open Systems Interconnection – Connection-oriented protocol for the Association Control Service Element. Protocol specification*

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

IETF RFC 793, *Transmission Control Protocol*, available at <<http://www.ietf.org>>

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

IETF RFC 1117, *Internet Numbers*, available at <<http://www.ietf.org>>

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

IETF RFC 1643, *Definitions of Managed Objects for the Ethernet-like Interface Types*, available at <<http://www.ietf.org>>

IETF RFC 2236, *Internet Group Management Protocol, Version 2*, available at <<http://www.ietf.org>>

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

IETF RFC 2597, *Assured Forwarding PHB Group*, available at <<http://www.ietf.org>>

IETF RFC 3140, *Per Hop Behavior Identification Codes*, available at <<http://www.ietf.org>>

IETF RFC 3246, *An Expedited Forwarding PHB (Per-Hop Behavior)*, available at <<http://www.ietf.org>>

IETF RFC 4541, *Considerations for Internet Group Management Protocol (IGMP) and Multicast Listener Discovery (MLD) Snooping Switches*, available at <<http://www.ietf.org>>

³ To be published.

IETF RFC 4594, *Configuration Guidelines for DiffServ Service Classes*, available at <<http://www.ietf.org>>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 1: Common Industrial Protocol (CIPTM) – Edition 3.5, November 2008*, available at <<http://www.odva.org>>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 2: EtherNet/IP™ Adaptation of CIP – Edition 1.6, November 2008*, available at <<http://www.odva.org>>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 3: DeviceNet™ Adaptation of CIP – Edition 1.6, November 2008*, available at <<http://www.odva.org>>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 4: ControlNet™ Adaptation of CIP – Edition 1.2, April 2008*, available at <<http://www.odva.org>>

ODVA: THE CIP NETWORKS LIBRARY – *Volume 5: CIP Safety™ – Edition 2.2, November 2008*, available at <<http://www.odva.org>>

IECNORM.COM: Click to view the full PDF of IEC 61158-6-2:2010

SOMMAIRE

AVANT-PROPOS	277
INTRODUCTION	279
1 Domaine d'application	280
1.1 Généralités	280
1.2 Spécifications	281
1.3 Conformité	281
2 Références normatives	281
3 Termes, définitions, symboles, abréviations et conventions	282
3.1 Termes et définitions issus d'autres normes ISO/CEI	282
3.2 Termes et définitions issus de la CEI 61158-5-2	284
3.3 Termes et définitions supplémentaires	284
3.4 Abréviations et symboles	291
3.5 Conventions	291
3.6 Conventions utilisées dans les diagrammes d'états	296
4 Syntaxe abstraite	299
4.1 Syntaxe abstraite des PDU de la FAL	299
4.2 Spécification de syntaxe abstraite des données	405
4.3 Syntaxe abstraite d'encapsulation	410
5 Syntaxe de transfert	427
5.1 Codage compact	427
5.2 Rapport relatif au type de données	435
6 Structure des diagrammes d'états de protocole de FAL	443
7 Diagramme d'état de contexte AP	443
7.1 Vue d'ensemble	443
7.2 Diagramme d'état d'objet Connection	443
8 Diagramme protocolaire de services de la FAL (FSPM)	456
8.1 Généralités	456
8.2 Définitions des primitives	456
8.3 Paramètres des primitives	461
8.4 Diagrammes d'états FSPM	462
9 Diagrammes protocolaires de relation d'applications (ARPM)	463
9.1 Généralités	463
9.2 ARPM sans connexion (UCMM)	463
9.3 Diagrammes ARP orientés connexion (transports)	478
10 Diagramme protocolaire de mise en correspondance DLL 1 (DMPM 1)	592
10.1 Généralités	592
10.2 Producteur de liaisons	593
10.3 Consommateur de liaisons	594
10.4 Définitions des primitives	594
10.5 Diagramme d'états DMPM	596
10.6 Sélection des services de couche liaison de données	598
11 Diagramme protocolaire de mise en correspondance DLL 2 (DMPM 2)	598
11.1 Généralités	598

11.2 Mise en correspondance des PDU de l'UCMM	599
11.3 Mise en correspondance des PDU des classes de transport 0 et 1	604
11.4 Mise en correspondance des PDU des classes de transport 2 et 3	606
11.5 Mise en correspondance des classes de transport 4 à 6	606
11.6 Utilisation de l'IGMP	606
11.7 Qualité de service (QoS) pour les messages CP 2/2	608
11.8 Gestion d'une session d'encapsulation	611
12 Diagramme protocolaire de mise en correspondance DLL 3 (DMPM 3)	612
Bibliographie	613

Figure 1 – Format et termes du tableau d'attributs	292
Figure 2 – Paramètre de demande/réponse de service	293
Figure 3 – Exemple d'un STD	297
Figure 4 – Paramètres de connexion réseau	320
Figure 5 – Temps de top d'horloge	323
Figure 6 – Temporisation d'établissement de connexion	327
Figure 7 – AttributTransport Class_Trigger	364
Figure 8 – Format de l'attribut CP2/3_initial_comm_characteristics	368
Figure 9 – Type de segment	377
Figure 10 – Segment de port	378
Figure 11 – Codage de segment logique	381
Figure 12 – Segment de réseau étendu	386
Figure 13 – Message d'encapsulation	411
Figure 14 – Règles de placement de bits associées au codage compact de FixedLengthBitString	431
Figure 15 – Exemple de codage compact d'un FixedLengthBitString SWORD	432
Figure 16 – Exemple de codage compact d'un FixedLengthBitString WORD	432
Figure 17 – Exemple de codage compact d'un FixedLengthBitString DWORD	432
Figure 18 – Exemple de codage compact d'un FixedLengthBitString LWORD	433
Figure 19 – Exemple 2 de codage formel d'une spécification de type structure	438
Figure 20 – Exemple de codage abrégé d'une spécification de type structure	439
Figure 21 – Exemple 1 de codage formel d'une spécification de type array	440
Figure 22 – Exemple 2 de codage formel d'une spécification de type array	441
Figure 23 – Exemple 1 de codage abrégé d'une spécification de type array	442
Figure 24 – Exemple 2 de codage abrégé d'une spécification de type array	443
Figure 25 – Diagramme de transition d'états d'objet Connection "I/O"	444
Figure 26 – Diagramme de transition d'états d'un objet Connection "Bridged"	450
Figure 27 – Diagramme de transitions d'états d'un objet Connection "Explicit Messaging"	453
Figure 28 – Diagramme de transitions d'états de client UCMM	466
Figure 29 – Diagramme de transitions d'états d'un serveur UCMM haut de gamme	470
Figure 30 – Diagramme de transitions d'états pour serveur UCMM bas de gamme	473

Figure 31 – Diagramme de séquence pour un UCMM avec un seul message en cours.....	475
Figure 32 – Diagramme de séquence pour un UCMM avec plusieurs messages en cours	477
Figure 33 – Tampon des TPDU.....	479
Figure 34 – Diagramme de flots de données utilisant une classe de transport client 0 et une classe de transport serveur 0	482
Figure 35 – Diagramme de séquence du transfert de données utilisant la classe de transport 0	483
Figure 36 – STD client de classe 0	485
Figure 37 – STD serveur de classe 0	487
Figure 38 – Diagramme de flots de données utilisant la classe de transport client 1 et la classe de transport serveur 1	488
Figure 39 – Diagramme de séquence de transfert de données utilisant la classe de transport client 1 et la classe de transport serveur 1	489
Figure 40 – STD client de classe 1	491
Figure 41 – STD serveur de classe 1	494
Figure 42 – Diagramme de flots de données utilisant la classe de transport client 2 et la classe de transport serveur 2	497
Figure 43 – Diagramme de transfert de données utilisant la classe de transport client 2 et la classe de transport serveur 2 sans données retournées.....	499
Figure 44 – Diagramme de séquence de transfert de données utilisant la classe de transport client 2 et la classe de transport serveur 2 avec données retournées.....	501
Figure 45 – STD client de classe 2	503
Figure 46 – STD serveur de classe 2	507
Figure 47 – Diagramme de flots de données utilisant la classe de transport client 3 et la classe de transport serveur 3	510
Figure 48 – Diagramme de séquence de transfert de données utilisant la classe de transport client 3 et la classe de transport serveur 3 sans données retournées.....	513
Figure 49 – Diagramme de séquence de transfert de données utilisant la classe de transport client 3 et la classe de transport serveur 3 avec données retournées.....	515
Figure 50 – STD client de classe 3	517
Figure 51 – STD serveur de classe 3	521
Figure 52 – Diagramme de flots de données utilisant les classes de transport 4 et 5	524
Figure 53 – Diagramme de séquences d'échange de messages utilisant les classes de transport 4 et 5	526
Figure 54 – Diagramme de séquences des messages s'écrasant les uns les autres.....	527
Figure 55 – Diagramme de séquences d'échange de messages placés en file d'attente utilisant les classes de transport 4 et 5	529
Figure 56 – Diagramme de séquences de répétitions de tentative utilisant les classes de transport 4 et 5	530
Figure 57 – Diagramme de séquences de trafic au repos utilisant les classes de transport 4 et 5	531
Figure 58 – Structure de base pour les classes 4 et 5	533
Figure 59 – Structure de base pour la classe 6	534
Figure 60 – STD général des classes 4 à 6.....	535

Figure 61 – STD expéditeur de classe 4	539
Figure 62 – STD destinataire de classe 4	543
Figure 63 – Diagramme de séquences de trois fragments utilisant la classe de transport 5	548
Figure 64 – Diagramme de séquences de fragmentation avec répétitions de tentative utilisant les classes de transport et 5	549
Figure 65 – Diagramme de séquences de deux fragments utilisant la classe de transport 5	550
Figure 66 – Diagramme de séquences de message abandonné utilisant la classe de transport 5	552
Figure 67 – STD expéditeur de classe 5	554
Figure 68 – STD destinataire de classe 5	560
Figure 69 – Diagramme de flot de données pour la classe de transport 6	566
Figure 70 – Diagramme de séquences d'échange de messages utilisant la classe de transport 6	568
Figure 71 – Diagramme de répétitions de tentative utilisant la classe de transport 6	569
Figure 72 – Diagramme de répétitions de trafic au repos utilisant la classe de transport 6	570
Figure 73 – Diagramme de séquences d'une demande écrasant en écriture null	572
Figure 74 – Diagramme de séquences d'une réponse écrasant en écriture un ACK de null	573
Figure 75 – Diagramme de séquences de trois fragments utilisant la classe de transport 6	575
Figure 76 – Diagramme de séquences de fragmentation avec répétitions de tentative utilisant la classe de transport 6	576
Figure 77 – Diagramme de séquences de deux fragments utilisant la classe de transport 6	577
Figure 78 – Diagramme de séquences de séquence fragmentée abandonnée utilisant la classe de transport 6	579
Figure 79 – STD client de classe 6	581
Figure 80 – STD serveur de classe 6	586
Figure 81 – Diagramme de flot de données pour producteur et un consommateur de données	593
Figure 82 – Diagramme de transitions d'états pour un producteur de liaisons	597
Figure 83 – Diagramme de transitions d'états pour un consommateur de liaisons	598
Figure 84 – Champ DS dans l'en-tête IP	609
Figure 85 – Trame étiquetée IEEE 802.1Q	610
Tableau 1 – Règles relatives au service de réponse Get_Attribute_All	293
Tableau 2 – Exemples de données de réponse spécifiques à un service/objet de niveau classe pour Get_Attribute_All	294
Tableau 3 – Exemple de méthode de matrice de données Get_Attribute_All	294
Tableau 4 – Règles relatives au service de demande Set_Attribute_All	295
Tableau 5 – Exemple de méthode d'ordonnancement d'attributs Set_Attribute_All	295
Tableau 6 – Exemple de méthode de matrice de données Set_Attribute_All	296
Tableau 7 – Format d'une matrice d'événements d'états	298

Tableau 8 – Exemple de matrice d'événements d'états	299
Tableau 9 – Format d'en-tête de l'UCMM_PDU	302
Tableau 10 – Codes de commande de l'UCMM	302
Tableau 11 – En-tête de transport de classe 0	303
Tableau 12 – En-tête de transport de classe 1	304
Tableau 13 – En-tête de transport de classe 2	304
Tableau 14 – En-tête de transport de classe 3	304
Tableau 15 – Format des en-têtes de classes 4 à 6	305
Tableau 16 – En-tête de données temps réel – propriétaire exclusif	306
Tableau 17 – En-tête de données temps réel – propriétaire redondant	307
Tableau 18 – Format de la demande Forward_Open	309
Tableau 19 – Format de la réponse Forward_Open_Good	310
Tableau 20 – Format de la réponse Forward_Open_Bad	310
Tableau 21 – Format de la demande Large_Forward_Open	312
Tableau 22 – Format de la réponse Large_Forward_Open_Good	312
Tableau 23 – Format de la réponse Large_Forward_Open_Bad	313
Tableau 24 – Format de la demande Forward_Close	314
Tableau 25 – Format de la réponse Forward_Close_Good	314
Tableau 26 – Format de la réponse Forward_Close_Bad	315
Tableau 27 – Format de demande Unconnected_Send	315
Tableau 28 – Format de la réponse Unconnected_Send_Good	316
Tableau 29 – Format de la réponse Unconnected_Send_Bad	317
Tableau 30 – Format de la demande Get_Connection_Data	317
Tableau 31 – Format de la réponse Get_Connection_Data	318
Tableau 32 – Format de la demande Search_Connection_Data	318
Tableau 33 – Format de la demande Get_Object_Owner	319
Tableau 34 – Format de la réponse Forward_Open_Good	319
Tableau 35 – Multiplicateur de temporisation	322
Tableau 36 – Unités de top d'horloge	323
Tableau 37 – Sélection de l'ID de connexion	328
Tableau 38 – Format de classe de transport, déclencheur et Is_Server	329
Tableau 39 – Format de MR_Request_Header	330
Tableau 40 – Format de MR_Response_Header	330
Tableau 41 – Structure du corps de Get_Attribute_All_ResponsePDU	331
Tableau 42 – Structure du corps de Set_Attribute_All_RequestPDU	331
Tableau 43 – Structure du corps de Get_Attribute_List_RequestPDU	331
Tableau 44 – Structure du corps de Get_Attribute_List_ResponsePDU	331
Tableau 45 – Structure du corps de Set_Attribute_List_RequestPDU	332
Tableau 46 – Structure du corps de Set_Attribute_List_ResponsePDU	332
Tableau 47 – Structure du corps de Reset_RequestPDU	332
Tableau 48 – Structure du corps de Reset_ResponsePDU	333

Tableau 49 – Structure du corps de Start_RequestPDU	333
Tableau 50 – Structure du corps de Start_ResponsePDU	333
Tableau 51 – Structure du corps de Stop_RequestPDU	333
Tableau 52 – Structure du corps de Stop_ResponsePDU.....	333
Tableau 53 – Structure du corps de Create_RequestPDU	334
Tableau 54 – Structure du corps de Create_ResponsePDU	334
Tableau 55 – Structure du corps de Delete_RequestPDU	334
Tableau 56 – Structure du corps de Delete_ResponsePDU.....	334
Tableau 57 – Structure du corps de Get_Attribute_Single_ResponsePDU.....	334
Tableau 58 – Structure du corps de Set_Attribute_Single_RequestPDU	335
Tableau 59 – Structure du corps de Set_Attribute_Single_ResponsePDU.....	335
Tableau 60 – Structure du corps de Find_Next_Object_Instance_RequestPDU.....	335
Tableau 61 – Structure du corps de Find_Next_Object_Instance_ResponsePDU.....	335
Tableau 62 – Structure du corps de Apply_Attributes_RequestPDU	336
Tableau 63 – Structure du corps de Apply_Attributes_ResponsePDU.....	336
Tableau 64 – Structure du corps de Save_RequestPDU	336
Tableau 65 – Structure du corps de Save_ResponsePDU	336
Tableau 66 – Structure du corps de Restore_RequestPDU.....	336
Tableau 67 – Structure du corps de Restore_ResponsePDU.....	337
Tableau 68 – Structure du corps de Group_Sync_RequestPDU	337
Tableau 69 – Structure du corps de Group_Sync_ResponsePDU.....	337
Tableau 70 – Attributs de classe d'objets Identity.....	337
Tableau 71 – Attributs d'instance d'objet Identity	337
Tableau 72 – Définitions de bits de l'objet Identity pour l'attribut d'instance de statut.....	339
Tableau 73 – Bits 4 – 7 de l'attribut d'instance de statut.....	339
Tableau 74 – Données de réponse spécifiques à un service/objet de niveau classe pour Get_Attribute_All	340
Tableau 75 – Données de réponse spécifiques à un service/objet de niveau instance pour Get_Attribute_All	340
Tableau 76 – Données de réponse modifiée spécifiques à un service/objet de niveau instance pour Get_Attribute_All	341
Tableau 77 – Paramètre spécifique à un objet pour Reset	341
Tableau 78 – Attributs de classe d'objets Message Router.....	342
Tableau 79 – Attributs d'instance d'objet Message Router	342
Tableau 80 – Données de réponse spécifiques à un service/objet de niveau classe pour Get_Attribute_All	342
Tableau 81 – Données de réponse spécifiques à un service/objet de niveau instance pour Get_Attribute_All	343
Tableau 82 – Attributs de classe d'objets Assembly	343
Tableau 83 – Attributs d'instance d'objet Assembly	343
Tableau 84 – Attributs de classe d'objets Acknowledge Handler	344
Tableau 85 – Attributs d'instance d'objet Acknowledge Handler	345

Tableau 86 – Structure du corps de l'Add_AckData_Path_RequestPDU	345
Tableau 87 – Structure du corps de Remove_AckData_Path_RequestPDU	346
Tableau 88 – Attributs d'instance d'objet Time Sync	346
Tableau 89 – Codage de ClockIdentity pour différentes mises en œuvre de réseau	350
Tableau 90 – Valeurs de ClockClass	350
Tableau 91 – Valeurs de TimeAcuracy	351
Tableau 92 – Valeurs de bit de TimePropertyFlags	351
Tableau 93 – Valeurs de TimeSource	352
Tableau 94 – Types de Clock	352
Tableau 95 – Attributs de classe d'objets Parameter	352
Tableau 96 – Valeurs des bits de Parameter Class Descriptor	353
Tableau 97 – Attributs d'instance d'objet Parameter	354
Tableau 98 – Sémantique de l'attribut d'instance Descriptor	355
Tableau 99 – Sémantique de Minimum Value et de Maximum Value	356
Tableau 100 – Attributs de formule de mise à l'échelle	357
Tableau 101 – Liaisons de mise à l'échelle	357
Tableau 102 – Structure du corps de Get_Enum_String_RequestPDU	358
Tableau 103 – Structure du corps de Get_Enum_String_ResponsePDU	358
Tableau 104 – Type de chaîne énumérée versus type de données de paramètres	358
Tableau 105 – Attributs de classe d'objets Connection Manager	359
Tableau 106 – Attributs d'instance d'objet Connection Manager	359
Tableau 107 – Attributs de classe d'objets Connection	360
Tableau 108 – Attributs d'instance d'objet Connection	361
Tableau 109 – Valeurs attribuées à l'attribut d'état	362
Tableau 110 – Valeurs attribuées à l'attribut instance_type	363
Tableau 111 – Valeurs possibles dans le bit Direction	365
Tableau 112 – Valeurs possibles dans les bits Production Trigger	365
Tableau 113 – Valeurs possibles dans les bits Transport Class	366
Tableau 114 – Attribut TransportClass_Trigger	367
Tableau 115 – Valeurs définies pour l'attribut CP2/3_produced_connection_id	367
Tableau 116 – Valeurs définies pour l'attribut CP2/3_consumed_connection_id	368
Tableau 117 – Valeurs pour le quartet Initial Production Characteristics	369
Tableau 118 – Valeurs pour le quartet Initial Consumption Characteristics	370
Tableau 119 – Valeurs de watchdog_timeout_action	373
Tableau 120 – Structure du corps de Connection_Bind_RequestPDU	375
Tableau 121 – Statut spécifique à un objet pour le service Connection_Bind	375
Tableau 122 – Structure du corps de Producing_Application_Lookup_RequestPDU	376
Tableau 123 – Structure du corps de Producing_Application_Lookup_ResponsePDU	376
Tableau 124 – Codes de statut du service Producing_Application_Lookup	376
Tableau 125 – Exemples de segments de port possibles	379
Tableau 126 – Exemples d'adresses de liaison TCP/IP	380

Tableau 127 – Format du segment "Electronic Key"	383
Tableau 128 – Exemples de segments logiques	383
Tableau 129 – Segments de réseau	384
Tableau 130 – Définitions des sous-types étendus	386
Tableau 131 – Segment de données	387
Tableau 132 – Segment de l'ANSI_Extended_Symbol	387
Tableau 133 – Catégories d'adressage	388
Tableau 134 – Plages des ID de code de classe	388
Tableau 135 – Plages des ID d'attribut	389
Tableau 136 – Plages de codes de service	389
Tableau 137 – Codes de classe	390
Tableau 138 – Attributs de classe réservés pour toutes les définitions de classes d'objets	390
Tableau 139 – Liste des services communs	391
Tableau 140 – Liste des services spécifiques à un objet Acknowledge Handler	392
Tableau 141 – Liste des services spécifiques à un objet Parameter	392
Tableau 142 – Services spécifiques au Connection Manager	392
Tableau 143 – Services spécifiques à l'objet Connection	393
Tableau 144 – Numérotation du type de dispositif	393
Tableau 145 – Codes d'erreur d'une demande de service de Connection Manager	394
Tableau 146 – Codes de statut général	400
Tableau 147 – Codes de statut d'objet Identity	403
Tableau 148 – En-tête d'encapsulation	411
Tableau 149 – Codes de commande d'encapsulation	412
Tableau 150 – Codes de statut d'encapsulation	413
Tableau 151 – Fanions d'options	413
Tableau 152 – En-tête d'encapsulation de demande Nop	414
Tableau 153 – En-tête d'encapsulation de demande RegisterSession	414
Tableau 154 – Partie "données" de la demande RegisterSession	414
Tableau 155 – Fanions d'options	415
Tableau 156 – En-tête d'encapsulation de réponse RegisterSession	415
Tableau 157 – Partie "données" de la réponse RegisterSession	415
Tableau 158 – En-tête d'encapsulation de demande UnRegisterSession	416
Tableau 159 – En-tête d'encapsulation de demande ListServices	417
Tableau 160 – En-tête d'encapsulation de réponse ListServices	417
Tableau 161 – Partie "données" de la réponse ListServices	417
Tableau 162 – Codes de type de services	418
Tableau 163 – Fanions de capacité de communications	418
Tableau 164 – En-tête d'encapsulation de demande ListIdentity	419
Tableau 165 – En-tête d'encapsulation de réponse ListIdentity	419
Tableau 166 – Partie "données" de la réponse ListIdentity	420

Tableau 167 – En-tête d'encapsulation de demande ListInterfaces	420
Tableau 168 – En-tête d'encapsulation de réponse ListInterfaces	421
Tableau 169 – En-tête d'encapsulation de demande SendRRData	421
Tableau 170 – Partie "données" de la demande SendRRData	422
Tableau 171 – En-tête d'encapsulation de réponse SendRRData	422
Tableau 172 – En-tête d'encapsulation de demande SendUnitData	423
Tableau 173 – Partie "données" de la demande SendUnitData	423
Tableau 174 – Format commun des paquets	423
Tableau 175 – Structure des éléments adresse et donnée	424
Tableau 176 – ID de type d'adresse	424
Tableau 177 – ID de type de donnée	424
Tableau 178 – Type d'adresse "null"	425
Tableau 179 – Type d'adresse "connected"	425
Tableau 180 – Type d'adresse "sequenced"	425
Tableau 181 – Type de données UCMM	425
Tableau 182 – Type de données connected	426
Tableau 183 – Éléments Sockaddr Info	426
Tableau 184 – Codage BOOLEAN	428
Tableau 185 – Exemple de codage compact d'une valeur BOOL	428
Tableau 186 – Codage des valeurs SignedInteger	428
Tableau 187 – Exemple de codage compact d'une valeur SignedInteger	428
Tableau 188 – Valeurs UnsignedInteger	428
Tableau 189 – Exemple de codage compact d'un UnsignedInteger	429
Tableau 190 – Valeurs FixedLengthReal	429
Tableau 191 – Exemple de codage compact d'une valeur REAL	429
Tableau 192 – Exemple de codage compact d'une valeur LREAL	429
Tableau 193 – Valeurs FixedLengthReal	429
Tableau 194 – Valeur STRING	430
Tableau 195 – Valeur STRING2	430
Tableau 196 – Valeur STRINGN	430
Tableau 197 – Valeur SHORT_STRING	430
Tableau 198 – Exemple de codage compact d'une valeur STRING	431
Tableau 199 – Exemple de codage compact d'une valeur STRING2	431
Tableau 200 – Type SHORT_STRING	431
Tableau 201 – Exemple de codage compact d'un ARRAY à une seule dimension	433
Tableau 202 – Exemple de codage compact d'un ARRAY à plusieurs dimensions	434
Tableau 203 – Exemple de codage compact d'une STRUCTURE	435
Tableau 204 – Codes d'identification et descriptions de types de données élémentaires	436
Tableau 205 – Exemple 1 de codage formel d'une spécification de type structure	437
Tableau 206 – Matrice d'événements d'états d'objet "I/O Connection"	445
Tableau 207 – Matrice d'événements d'états d'une connexion "Bridged "	451

Tableau 208 – Matrice d'événements d'états d'une connexion "Explicit Messaging"	454
Tableau 209 – Primitives émises par l'utilisateur de la FAL vers le diagramme FSPM	457
Tableau 210 – Primitives émises par l'utilisateur de la FAL vers le diagramme FSPM	458
Tableau 211 – Primitives émises par le diagramme FSPM vers l'utilisateur de la FAL	460
Tableau 212 – Paramètres utilisés avec les primitives échangées entre l'utilisateur de la FAL et le diagramme FSPM	462
Tableau 213 – Primitives émises par le diagramme FSPM vers l'ARPM	464
Tableau 214 – Primitives émises par de l'ARPM vers le diagramme FSPM	464
Tableau 215 – Paramètres utilisés avec les primitives échangées entre le diagramme FSPM et l'ARPM	465
Tableau 216 – États du client UCMM	465
Tableau 217 – Matrice d'événements d'états du client UCMM	467
Tableau 218 – États du serveur UCMM haut de gamme	468
Tableau 219 – Matrice d'événements d'états du serveur UCMM haut de gamme	471
Tableau 220 – États du serveur UCMM bas de gamme	472
Tableau 221 – Matrice d'événements d'états du serveur UCMM bas de gamme	474
Tableau 222 – Notification	480
Tableau 223 – Classes de transport	480
Tableau 224 – Primitives émises par le diagramme FSPM vers l'ARPM	481
Tableau 225 – Primitives émises par de l'ARPM vers le diagramme FSPM	481
Tableau 226 – Paramètres utilisés avec les primitives échangées entre le diagramme FSPM et l'ARPM	481
Tableau 227 – États du client de transport de classe 0	484
Tableau 228 – SEM client de classe 0	485
Tableau 229 – États du serveur de transport de classe 0	486
Tableau 230 – SEM serveur de classe 0	487
Tableau 231 – États du client de transport de classe 1	491
Tableau 232 – SEM client de classe 1	492
Tableau 233 – États du serveur de transport de classe 1	493
Tableau 234 – SEM serveur de classe 1	495
Tableau 235 – États du client de transport de classe 2	502
Tableau 236 – SEM client de classe 2	504
Tableau 237 – États du serveur de transport de classe 2	505
Tableau 238 – SEM serveur de classe 2	507
Tableau 239 – États du client de transport de classe 3	516
Tableau 240 – SEM client de classe 3	518
Tableau 241 – États du serveur de transport de classe 3	519
Tableau 242 – SEM serveur de classe 3	522
Tableau 243 – Événements d'écriture et de déclenchement dans un transport des classes 4 et 5	525
Tableau 244 – États communs pour les classes de transport 4 à 6	535
Tableau 245 – SEM générale des classes 4 à 6	536

Tableau 246 – États de l'expéditeur de transport de classe 4	538
Tableau 247 – SEM expéditeur de classe 4	540
Tableau 248 – États du destinataire de transport de classe 4	542
Tableau 249 – SEM destinataire de classe 4.....	545
Tableau 250 – États de l'expéditeur de transport de classe 5.....	553
Tableau 251 – SEM expéditeur de classe 5	556
Tableau 252 – États du destinataire de transport de classe 5	558
Tableau 253 – SEM destinataire de classe 5.....	561
Tableau 254 – États du client de transport de classe 6	580
Tableau 255 – Matrice d'événements d'états de client de classe 6.....	582
Tableau 256 – États du serveur de transport de classe 6.....	584
Tableau 257 – SEM serveur de classe 6	588
Tableau 258 – Primitives émises par l'ARPM vers le diagramme DMPM.....	594
Tableau 259 – Primitives émises par le DMPM vers l'ARPM.....	594
Tableau 260 – Paramètres utilisés avec les primitives échangées entre l'ARPM et le DMPM	594
Tableau 261 – Primitives échangées entre la couche liaison de données et le DMPM.....	595
Tableau 262 – Paramètres utilisés avec les primitives échangées entre DMPM et la couche liaison de données.....	596
Tableau 263 – États du protecteur de liaisons.....	596
Tableau 264 – Matrice d'événements d'états du producteur de liaisons	597
Tableau 265 – États du consommateur de liaisons	597
Tableau 266 – Matrice d'événements d'états du consommateur de liaisons	598
Tableau 267 – Sélection de l'ID de connexion réseau	600
Tableau 268 – Exemples d'attributions de multidiffusion	604
Tableau 269 – Format des données UDP pour la classe 0 et la classe 1.....	605
Tableau 270 – Mise en correspondance des DSCP par défaut et de l'IEEE 802.1D.....	610

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

**RÉSEAUX DE COMMUNICATION INDUSTRIELS –
SPÉCIFICATIONS DES BUS DE TERRAIN –****Partie 6-2: Spécification de protocole de la couche application –
Éléments de Type 2**

AVANT-PROPOS

- 1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
- 2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
- 3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications; la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
- 4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
- 5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
- 6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
- 7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
- 8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
- 9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de propriété intellectuelle ou de droits analogues. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de propriété et de ne pas avoir signalé leur existence.

NOTE 1 L'utilisation de certains des types de protocoles associés est limitée par les détenteurs de leurs droits de propriété intellectuelle. Dans tous les cas, l'engagement à un abandon limité des droits de propriété intellectuelle pris par les détenteurs de ces droits permet d'utiliser un type particulier de protocole de couche liaison de données avec des protocoles de couche physique et de couche application dans des combinaisons de types telles que spécifiées de façon explicite dans les parties profil. L'utilisation des divers types de protocoles dans d'autres combinaisons peut exiger la permission donnée par les détenteurs respectifs de leurs droits de propriété intellectuelle.

La Norme internationale CEI 61158-6-2 a été établie par le sous-comité 65C: Réseaux de communication industriels, du comité d'études 65 de la CEI: Mesure, commande et automation dans les processus industriels.

Cette deuxième édition annule et remplace la première édition, parue en 2007. Cette édition constitue une révision technique.

Les principales modifications par rapport à l'édition précédente sont énumérées ci-dessous:

- Article 2 et Bibliographie: mise à jour des références normatives et bibliographiques;
- paragraphe 3.4: mise à jour des abréviations;
- paragraphe 4.1.2.1: liste de mise à jour pour les PDU de demande/réponse de service (ASE/objets Time Sync et Parameter);
- paragraphe 4.1.8.6: mise à jour d'ASE/objet Time Sync;
- nouveau paragraphe 4.1.8.7: nouvel ASE/objet Parameter;
- paragraphe 4.1.9.4: mise à jour mineure d'un paramètre commun;
- paragraphe 4.1.10.2: mise à jour/ajout de codes d'objet et de services pour les ASE Time Sync et Parameter;
- paragraphe 4.1.11: mises à jour mineures de codes d'erreur;
- nouveau paragraphe 11.7: nouvelle spécification de QoS.

La présente version bilingue (2012-06) correspond à la version anglaise monolingue publiée en 2010-08.

Le texte anglais de cette norme est issu des documents 65C/607/FDIS et 65C/621/RVD.

Le rapport de vote 65C/621/RVD donne toute information sur le vote ayant abouti à l'approbation de cette norme.

La version française de cette norme n'a pas été soumise au vote.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 61158, présentées sous le titre général *Réseaux de communication industriels – Spécifications de bus de terrain*, peut être consultée sur le site web de la CEI.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "<http://webstore.iec.ch>" dans les données relatives à la publication recherchée. À cette date, la publication sera:

- reconduite;
- supprimée;
- remplacée par une édition révisée, ou
- amendée.

NOTE 2 La révision de la présente norme sera synchronisée avec les autres parties de la série CEI 61158.

INTRODUCTION

La présente partie de la CEI 61158 est l'une d'une série produite pour faciliter l'interconnexion de composants d'un système d'automatisation. Elle est apparentée à d'autres normes de la série telle que définie par le modèle de référence des bus de terrain à trois couches décrit dans le rapport IEC/TR 61158-1:2010.

Le protocole application fournit le service application en utilisant les services disponibles de la couche liaison de données ou autre couche immédiatement inférieure. Le but principal de la présente norme est de fournir un ensemble de règles pour la communication exprimées en termes des procédures devant être accomplies par des entités d'application (AE) d'homologues au moment de la communication. Ces règles pour la communication visent à fournir une base solide pour le développement afin de servir une diversité de besoins:

- comme un guide pour les réalisateurs et les concepteurs;
- pour une utilisation dans les essais et achats d'équipements;
- comme partie intégrante d'un accord pour l'admission de systèmes dans l'environnement des systèmes ouverts;
- comme affinement pour la compréhension de communications à temps critiques au sein de l'OSI.

La présente norme est concernée, en particulier, par la communication et l'interfonctionnement des capteurs, des effecteurs et autres dispositifs d'automatisation. L'utilisation de la présente norme conjointement à d'autres normes positionnées dans les modèles de référence de l'OSI ou de bus de terrain permet à n'importe quelle combinaison de systèmes autrement incompatibles de fonctionner.

RÉSEAUX DE COMMUNICATION INDUSTRIELS – SPÉCIFICATIONS DES BUS DE TERRAIN –

Partie 6-2: Spécification de protocole de la couche application – Eléments de Type 2

1 Domaine d'application

1.1 Généralités

La Couche application de bus de terrain (FAL¹) fournit aux programmes d'utilisateur un moyen d'accéder à l'environnement de communication du bus de terrain. À cet égard, la FAL peut être vue comme une «fenêtre entre des programmes d'application correspondants».

La présente norme fournit les éléments communs pour les communications de messagerie de base à temps critique et à temps non critique entre des programmes d'application dans un environnement d'automatisation et le matériel spécifique au bus de terrain de Type 2. Le terme "à temps critique" sert à représenter la présence d'une fenêtre temporelle, dans les limites de laquelle une ou plusieurs actions spécifiées sont tenues d'être parachevées avec un certain niveau défini de certitude. Le manquement à parachever les actions spécifiées dans les limites de la fenêtre temporelle risque d'entraîner la défaillance des applications qui demandent ces actions, avec le risque concomitant pour l'équipement, l'installation et éventuellement pour la vie humaine.

La présente norme spécifie les interactions entre les applications distantes et définit le comportement visible de l'extérieur fourni par la couche application de bus de terrain de Type 2 en termes

- a) de la syntaxe abstraite formelle définissant les unités de données de protocole de couche application acheminées entre les entités d'application engagées dans une communication;
- b) de la syntaxe de transfert définissant les règles de codage qui sont appliquées aux unités de données de protocole de couche application;
- c) du diagramme d'états de contexte application définissant le comportement de service application visible entre des entités d'application engagées dans une communication;
- d) des diagrammes d'états de relation d'applications définissant le comportement de communication visible entre des entités d'application engagées dans une communication.

Le but de la présente norme est de définir le protocole fourni pour

- a) définir la représentation câblée des primitives de service définies dans la IEC 61158-5-2:2010, et
- b) définir le comportement visible de l'extérieur qui est associé à leur transfert.

La présente norme spécifie le protocole de la couche application des bus de terrain de Type 2, en conformité avec le Modèle de référence de base de l'OSI (ISO/CEI 7498-1) et la structure de la couche application de l'OSI (ISO/CEI 9545).

¹ FAL = *Fieldbus Application Layer*

1.2 Spécifications

L'objet principal de la présente norme est de spécifier la syntaxe et le comportement du protocole de couche application qui achemine les services de couche application définis dans la IEC 61158-5-2:2010.

Un objectif secondaire est de fournir des trajets de migration à partir de protocoles de communications industrielles préexistants. C'est ce dernier objectif qui donne naissance à la diversité des protocoles normalisés dans les diverses parties IEC 61158-6-x.

1.3 Conformité

La présente norme ne spécifie aucune mise en œuvre ou aucun produit individuels, de même qu'elle ne restreint nullement les mises en œuvre des entités de couche application dans les systèmes d'automation industriels. La conformité est obtenue par la mise en œuvre de cette spécification de protocole de couche application.

2 Références normatives

Les documents de référence suivants sont indispensables pour l'application du présent document. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

IEC 61158-3-2:2007, *Industrial communication networks – Fieldbus specifications – Part 3-2: Data-link layer service definition – Type 2 elements* (disponible uniquement en anglais)²

IEC 61158-4-2:2010³, *Industrial communication networks – Fieldbus specifications – Part 4-2: Data-link layer protocol specification – Type 2 elements* (disponible uniquement en anglais)

IEC 61158-5-2:2010³, *Industrial communication networks – Fieldbus specifications – Part 5-2: Application layer service definition – Type 2 elements* (disponible uniquement en anglais)

IEC 61588:2009, *Precision clock synchronization protocol for networked measurement and control systems* (disponible uniquement en anglais)

IEC 61784-3-2, *Industrial communication networks – Profiles – Part 3-2: Functional safety fieldbuses – Additional specifications for CPF 2* (disponible uniquement en anglais)

CEI 62026-3:2008, *Appareillage à basse tension – Interfaces appareil de commande-appareil (CDI) – Partie 3: DeviceNet*

ISO/CEI 7498-1, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base: Le modèle de base*

ISO/IEC 8802-3, , *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications* (disponible uniquement en anglais)

² Les publications monolingues des séries IEC 61158 et IEC 61784 sont actuellement en cours de traduction.

³ A publier..

ISO/CEI 8824 (toutes les parties), *Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1)*

ISO/CEI 8825-1, *Technologies de l'information – Règles de codage ASN.1 : Spécification des règles de codage de base (BER), des règles de codage canoniques (CER) et des règles de codage distinctives (DER)*

ISO/CEI 9545, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Structure de la couche application*

ISO/CEI 10646, *Technologies de l'information – Jeu universel de caractères codés sur plusieurs octets (JUC)*

ISO/CEI 10731, *Technologies de l'information – Interconnexion de systèmes ouverts (OSI) – Modèle de référence de base – Conventions pour la définition des services OSI*

ISO 11898:1993⁴, *Véhicules routiers – Échange d'information numérique – Gestionnaire de réseau de communication à vitesse élevée (CAN)*

IEEE 802.1D, *IEEE standard for local and metropolitan area networks – Media Access Control (MAC) bridges*, disponible à l'adresse <http://www.ieee.org>

IEEE 802.1Q, *IEEE standard for local and metropolitan area networks – Virtual bridged local area networks*, disponible à l'adresse <http://www.ieee.org>

IETF RFC 791, *Internet Protocol*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1035, *Domain Names – Implementation and Specification*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1122, *Requirements for Internet Hosts – Communication Layers*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 1759, *Printer MIB*, disponible à l'adresse <http://www.ietf.org>

IETF RFC 2475, *An Architecture for Differentiated Services*, disponible à l'adresse <http://www.ietf.org>

3 Termes, définitions, symboles, abréviations et conventions

Pour les besoins du présent document, les termes, définitions, symboles, abréviations et conventions suivants s'appliquent.

3.1 Termes et définitions issus d'autres normes ISO/CEI

3.1.1 Termes et définitions issus de l'ISO/CEI 7498-1

- a) syntaxe abstraite
- b) entité d'application
- c) processus d'application

⁴ Une édition plus récente de cette norme existe, mais seule l'édition citée s'applique.

- d) unité de donnée de protocole application
- e) élément de service application
- f) invocation d'entité d'application
- g) invocation de processus application
- h) transaction d'application
- i) contexte de présentation
- j) système ouvert réel
- k) syntaxe de transfert

3.1.2 Termes et définitions issus de l'ISO/CEI 9545

- a) association d'applications
- b) contexte d'application
- c) nom de contexte d'application
- d) invocation d'entité d'application
- e) type d'entité d'application
- f) invocation de processus d'application
- g) type de processus d'application
- h) élément de service d'application
- i) élément de service de contrôle d'application

3.1.3 Termes et définitions issus de l'ISO/CEI 8824

- a) object identifier (Identificateur d'objet)
- b) type
- c) valeur
- d) type simple
- e) type structuré
- f) type de composant
- g) tag ("étiquette")
- h) type Boolean (booléen)
- i) true ("vrai")
- j) false ("faux")
- k) type integer ("entier")
- l) type bitstring ("chaîne binaire")
- m) type octetstring ("chaîne d'octets")
- n) type null ("vide")
- o) type séquence
- p) séquence de type
- q) type choice ("choix")
- r) type tagged ("étiqueté")
- s) any type ("n'importe quel type")
- t) module

u) production

3.1.4 Termes et définitions issus de l'ISO/CEI 8825-1

- a) codage (d'une valeur de donnée)
- b) valeur de donnée
- c) octets d'identificateur (le singulier est utilisé dans la présente norme)
- d) octet(s) de longueur (le singulier et le pluriel sont utilisés dans la présente norme)
- e) octets de contenu

3.2 Termes et définitions issus de la CEI 61158-5-2

- a) relation d'applications
- b) client
- c) homologue/pair
- d) serveur

3.3 Termes et définitions supplémentaires

3.3.1

allouer

prendre une ressource dans une zone commune et affecter la ressource en question à l'usage exclusif d'une entité spécifique

3.3.2

application

fonction ou structure de données pour laquelle des données sont consommées ou produites

3.3.3

objets d'application

classes d'objets multiples qui gèrent et assurent un échange de messages pendant le mode exécution à travers le réseau et à l'intérieur du dispositif de réseau

3.3.4

attribut

description d'une caractéristique ou fonction visible de l'extérieur d'un objet

NOTE Les attributs d'un objet contiennent de l'information relative à des parties variables d'un objet. Typiquement, ils fournissent des informations de statut ou régissent le fonctionnement d'un objet. Des attributs peuvent aussi avoir une incidence sur le comportement d'un objet. Les attributs se répartissent en attributs de classes et attributs d'instances.

3.3.5

comportement

indication de la façon dont un objet réagit à des événements particuliers

3.3.6

horloge frontière

horloge avec plus d'un seul port PTP, chacun de ces ports PTP fournissant l'accès à un trajet de communication PTP distinct

NOTE Les horloges frontières sont utilisées pour éliminer les fluctuations produites par les routeurs et éléments de réseau similaires.

3.3.7**appelé**

utilisateur de service ou fournisseur de service qui reçoit une primitive "indication" ou une APDU de demande

3.3.8**appelant**

utilisateur de service ou fournisseur de service qui lance une primitive "indication" ou une APDU de demande

3.3.9**classe**

ensemble d'objets, qui représentent tous le même type de composant système

NOTE Une classe est une généralisation d'un objet; un modèle pour définir des variables et des méthodes. Tous les objets dans une classe ont une forme et un comportement identiques, mais contiennent en général des données différentes dans leurs attributs

3.3.10**attribut de classe**

attribut qui est partagé par tous les objets au sein de la même classe

3.3.11**code de classe**

identificateur unique attribué à chaque classe d'objets

3.3.12**service spécifique à une classe**

service défini par une classe d'objets particulière pour accomplir une fonction requise qui n'est pas accomplie par un service commun

NOTE Un objet spécifique à une classe est unique pour la classe d'objets qui le définit.

3.3.13**client**

- a) objet qui utilise les services d'un autre objet (serveur) pour accomplir une tâche
- b) initiateur d'un message auquel un serveur réagit

3.3.14**horloge**

dispositif donnant une mesure de l'écoulement du temps depuis une époque définie

NOTE Il y a deux types d'horloges dans la CEI 61588:2009: les horloges frontières et les horloges ordinaires

3.3.15**objets de communication**

composants qui gèrent et assurent l'échange de messages pendant le mode exécution à travers le réseau

EXEMPLES objet Connection Manager (gestionnaire de connexion), objet Unconnected Message Manager (UCMM, gestionnaire de message non connecté), objet Message Router (routeur de message).

3.3.16**connexion**

liaison logique entre deux objets d'application qui peuvent se trouver au sein du même dispositif ou dans des dispositifs différents

NOTE Les connexions sont soit point à point, soit multipoint.

3.3.17

ID de connexion

CID (*connection ID*)

identificateur affecté à une émission qui est associé à une connexion particulière entre producteurs et consommateurs, donnant un nom à un élément spécifique d'informations d'application

3.3.18

chemin de connexion

train d'octets qui définit l'objet d'application auquel une instance de connexion s'applique

3.3.19

point de connexion

tampon qui est représenté comme étant une sous-instance d'un objet Assembly (ensemble)

3.3.20

consommer

recevoir des données d'un producteur

3.3.21

consommateur

nœud ou puits qui reçoit des données d'un producteur

3.3.22

application consommatrice

application qui consomme des données

3.3.23

cyclique

répétitif d'une manière régulière

3.3.24

dispositif

équipement matériel physique relié à la liaison

NOTE Un dispositif peut contenir plus d'un nœud.

3.3.25

profil de dispositif

ensemble d'informations et de fonctionnalité dépendant du dispositif qui assure la cohérence entre des dispositifs similaires relevant du même type de dispositif

3.3.26

nœud d'extrémité

nœud producteur ou consommateur

3.3.27

point d'extrémité, point final

l'une des entités en communication impliquées dans une connexion

3.3.28

époque

temps de référence définissant l'origine d'une échelle de temps
[CEI 61588:2009]

3.3.29**erreur**

discordance entre une valeur ou un état calculé(e), observé(e) ou mesuré(e) et la valeur ou l'état spécifié(e) ou théoriquement correct(e)

3.3.30**trame**

synonyme déconseillé de DLPDU

3.3.31**horloge grand-mère**

horloge qui sert de source primaire de temps à laquelle toutes les autres (au sein d'un ensemble d'horloges CEI 61588) sont synchronisées en dernier ressort

3.3.32**instance**

occurrence physique réelle d'un objet au sein d'une classe identifiant un parmi plusieurs objets au sein la même classe d'objets

EXEMPLE "California" (La Californie) est une instance de la classe d'objets "state" (état).

NOTE Les termes "objet", "instance" et "instance d'objet" sont utilisés pour se référer à une instance spécifique.

3.3.33**attribut d'instance**

attribut qui est propre à une instance d'objet et n'est pas partagé par la classe d'objets

3.3.34**instancié**

objet qui a été créé dans un dispositif

3.3.35**interopérabilité**

capacité des entités de Couche Utilisateur à accomplir des opérations coordonnées et coopératives en utilisant les services de la FAL

3.3.36**Gardien (Keeper)**

objet chargé de distribuer les données de configuration d'une liaison à tous les nœuds de la liaison

3.3.37**petit boutiste**

décrit un modèle d'organisation de la mémoire qui stocke l'octet de poids faible à l'adresse la plus basse, ou pour le transfert, qui transfère en premier l'octet d'ordre le plus faible

3.3.38**Lpacket**

élément d'information d'application qui contient une taille, un octet de contrôle, une étiquette, et des données de liaison

NOTE Les couches liaison de données des homologues utilisent des Lpacket pour envoyer et recevoir des unités de données de service provenant des couches supérieures de la pile OSI.

3.3.39**informations de gestion**

informations accessibles du réseau qui prennent en charge la gestion de l'environnement du bus de terrain

3.3.40

horloge mère

horloge unique qui sert de source primaire de temps au sein de chaque région et qui, à son tour, se synchronisera à d'autres horloges mères et en dernier ressort à l'horloge grand-mère

NOTE Un système d'horloges CEI 61588 peut être segmenté en régions séparées par des horloges frontières.

3.3.41

membre

élément d'un attribut qui est structuré comme étant un élément d'une matrice

3.3.42

routeur de message (*Message Router*)

objet au sein d'un nœud qui distribue les demandes de messagerie vers les objets d'application appropriés

3.3.43

connexion multipoint

connexion d'un nœud à plusieurs autres nœuds

NOTE Les connexions multipoint permettent que les messages issus d'un seul producteur soient reçus par plusieurs nœuds consommateurs.

3.3.44

réseau

ensemble de nœuds reliés par un certain type de support de communication, notamment des répéteurs intermédiaires, ponts, routeurs et passerelles de couche inférieure

3.3.45

objet

représentation abstraite d'un composant particulier au sein d'un dispositif, habituellement un ensemble de données connexes (sous la forme de variables) et de méthodes (procédures) pour effectuer des opérations sur les données en question qui ont une interface et un comportement clairement définis

3.3.46

service spécifique à un objet

service propre à la classe d'objets qui le définit

3.3.47

horloge ordinaire

horloge CEI 61588 ayant un seul port PTP

3.3.48

émetteur

client chargé d'établir un chemin de communication vers une cible

3.3.49

connexion point à point

connexion qui existe entre exactement deux objets d'application

3.3.50

protocole de temps de précision

PTP (*Precision Time Protocol*)

nom utilisé pour le protocole de synchronisation du temps

3.3.51**produire**

envoyer des données devant être reçues par un consommateur

3.3.52**producteur**

nœud qui est chargé d'envoyer des données

3.3.53**message PTP**

message de synchronisation du temps CEI 61588

NOTE Il y a cinq types désignés de messages définis par la CEI 61588:2009: Sync, Delay_Req, Follow-up, Delay_Resp, et Management.

3.3.54**port PTP**

point d'accès logique pour les communications CEI 61588 à l'horloge contenant le port

3.3.55**récepteur**

utilisateur de service qui reçoit une primitive confirmée ou une primitive non confirmée, ou fournisseur de service qui reçoit une APDU confirmée ou une APDU non confirmée

3.3.56**ressource**

capacité de traitement ou d'informations d'un sous-système

3.3.57**expéditeur**

utilisateur de service qui envoie une primitive confirmée ou une primitive non confirmée, ou fournisseur de service qui envoie une APDU confirmée ou une APDU non confirmée

3.3.58**serveur**

- a) rôle d'un AREP (point d'extrémité de relation d'applications) dans lequel il retourne au client qui a lancé la demande une APDU confirmée de réponse de service
- b) objet qui fournit des services à un autre objet (client)

3.3.59**service**

opération ou fonction accomplie par un objet et/ou une classe d'objets à la demande d'un autre objet et/ou d'une autre classe d'objets

3.3.60**horloges synchronisées**

(avec une incertitude spécifiée) deux horloges qui ont la même époque et pour lesquelles les mesures de n'importe quel intervalle de temps par les horloges ne diffèrent pas plus de l'incertitude spécifiée

NOTE Les marqueurs temporels générés par deux horloges synchronisées pour le même événement ne différeront pas de plus de l'incertitude spécifiée.

3.3.61

Heure Système

temps absolu tel que défini par la synchronisation de CPF2 dans le contexte d'un système de temps distribué où tous les dispositifs ont une horloge locale qui est synchronisée avec une horloge mère commune

NOTE Dans le contexte de CPF2, System Time (Heure Système) est une valeur entière de 64 bits exprimée en unités de nanosecondes avec une valeur 0 correspondant à la date 1970-01-01 (c'est-à-dire 1^{er} janvier 1970).

3.3.62

cible

nœud d'extrémité auquel une connexion est établie

3.3.63

nœud temporaire

nœud transitoire

3.3.64

id de transaction (*transaction id*)

champ dans un en-tête UCMM qui met une réponse en correspondance avec la demande associée

3.3.65

Gestionnaire de messages non confirmés

UCMM (*Unconnected Message Manager*)

composant au sein d'un nœud qui émet et reçoit des messages explicites non connectés et les envoie directement à l'objet Message Router (routeur de message)

3.3.66

service non connecté

service de messagerie qui ne repose pas sur l'établissement d'une connexion entre les dispositifs avant d'autoriser des échanges d'informations

3.3.67

ID de vendeur (*vendor ID*)

identification de chaque fabricant/fournisseur de produit par un numéro unique

NOTE Les ID de vendeur sont attribués par ODVA, Inc.

3.4 Abréviations et symboles

ASCII	American Standard Code for Information Interchange (Code américain normalisé pour l'échange d'information)
CID	connection ID (ID de connexion)
DLL	data-link layer (Couche liaison de données)
DSCP	DiffServ Codepoint (point de code de service différencié)
IGMP	Internet Group Management Protocol (protocole de gestion de groupe internet, voir IETF RFC 1112, IETF RFC 2236)
IP	Internet Protocol (protocole internet, voir IETF RFC 791)
IPv4	Internet Protocol version 4 (protocole internet version 4, voir IETF RFC 791)
IPv6	Internet Protocol version 6 (protocole internet version 6, voir IETF RFC 791)
OSI	open systems interconnection (interconnexion des systèmes ouverts, voir ISO/CEI 7498-1)
PDU	protocol data unit (unité de données de protocole)
PHB	per-hop behavior (comportement par saut)
PTP	Precision Time Protocol (protocole de temps de précision) [CEI 61588:2009]
QoS	quality of service (qualité de service)
Rcv	recevoir
Rx	recevoir
SDU	service data unit (unité de donnée de service)
SEM	state event matrix (matrice d'événements d'états)
STD	state transition diagram (diagramme des transitions d'états), utilisé pour décrire le comportement d'objet
TCP	Terminal Control Protocol (protocole de contrôle de terminal, voir IETF RFC 793)
ToS	type of service (type de service)
TPDU	transport protocol data unit (unité de données de protocole de transport)
Tx	émettre
UDP	User Datagram Protocol (protocole datagramme d'utilisateur, voir IETF RFC 768)
Xmit	émettre
CM_API	actual packet interval (intervalle réel entre paquets)
O2T ou O⇒T	émetteur vers cible (paramètres de connexion)
CM_RPI	requested packet interval (intervalle requis entre paquets)
TUI	table unique identifier (identificateur unique de tableau)
T2O ou T⇒O	cible vers émetteur (paramètres de connexion)

3.5 Conventions

3.5.1 Concept général

La couche FAL est définie comme un jeu d'éléments ASE orientés objet. Chaque ASE est spécifié dans un paragraphe distinct. Chaque spécification d'ASE est constituée de trois parties: ses définitions de classe, ses services et sa spécification de protocole. Les deux premières parties sont contenues dans la CEI 61158-5-2:2010. La spécification de protocole pour chacun des ASE est définie dans la présente norme.

Les définitions de classe définissent les attributs pris en charge par chaque ASE. Les attributs sont accessibles à partir d'instances de la classe en utilisant les services d'ASE de

gestion ("Management") spécifiés dans la CEI 61158-5-2:2010. La spécification de services définit les services qui sont fournis par l'ASE.

La présente norme utilise les conventions descriptives données dans l'ISO/CEI 10731.

3.5.1.1 Spécification d'attribut

Les attributs sont définis dans un Tableau d'attributs utilisant le format et les termes définis à la Figure 1.

Attribute ID	Name	Data type	Semantics of values

Légende

Attribute ID	Identificateur d'attribut
Name	Nom
Data type	Type de données
Semantics of values	Sémantique des valeurs

Figure 1 – Format et termes du tableau d'attributs

Attribute ID doit être une valeur entière d'identification unique affectée à un attribut. Les plages valides pour les ID d'attribut doivent être celles spécifiées en 4.1.10.1.3. L'ID d'attribut doit identifier l'attribut particulier auquel il est fait accès.

Name doit se référer au nom affecté à l'attribut. Un attribut peut contenir des sous-éléments, qui peuvent aussi avoir des noms, comme c'est le cas avec le data type (type de données) **STRUCT of**. Le nom d'attribut doit être le nom qui apparaît le premier ou à la ligne supérieure dans la colonne de nom (la ligne qui contient l'ID d'attribut).

Chaque attribut doit se voir affecter un **Data type** qui doit être soit un type de donnée élémentaire, soit un type de donnée dérivé. Les types de donnée pour les attributs définis doivent être utilisés dans toutes les mises en œuvre.

NOTE Les types de données élémentaires sont définis dans la CEI 61158-5-2:2010.

Semantics of values (la sémantique des valeurs) doit spécifier la signification de la valeur (ou des valeurs) de l'attribut. Si cette information a besoin de plus d'espace qu'il ne peut s'en insérer dans le tableau, elle suivra immédiatement le Tableau Attribut de classe. Une référence adéquate à cette information sera incluse dans le Tableau Attribut de classe.

Si un Attribut de classe est *facultatif*, une valeur par défaut ou une méthode de traitement de cas spéciaux doit être définie afin que le Client (Demandeur) puisse traiter le message d'erreur qui se produit lors de l'accès aux objets qui choisissent de ne pas mettre en œuvre l'attribut de classe.

3.5.1.2 Services communs

3.5.1.2.1 Définitions des Service_PDU

Chaque service a des paramètres uniques pour la demande et la réponse. Les paramètres de demande et de réponse de service doivent être définis avec l'aide des tableaux de paramètres Request/Response (c'est-à-dire: Demande/Réponse) de service tels que définis à la Figure 2.

Name	Type	Semantics of values

Légende

Anglais	Français
Name	Nom
Type	Type
Semantics of values	Sémantique des valeurs

Figure 2 – Paramètre de demande/réponse de service

Name doit faire référence au nom donné au paramètre de demande/réponse de service.

Type doit spécifier le type de donnée du paramètre de demande/réponse de service.

Semantics of values (sémantiques des valeurs) doit spécifier la signification des valeurs du paramètre de demande/réponse de service (par exemple: «la valeur est en comptes en microsecondes.»).

3.5.1.2.2 Réponse Get_Attribute_All

3.5.1.2.2.1 Définition générale

Lorsque le service commun Get_Attribute_All est inclus dans la liste des services System/Object Management (c'est-à-dire: gestion de système/objet) pris en charge, la réponse **Get_Attribute_All** doit être détaillée pour cette classe: la séquence ou l'ordre des données retournées dans la Service_ResponsePDU doit être spécifié(e). Il existe trois façons dont il est permis de spécifier la Get_Attribute_All Service_ResponsePDU:

- énumérer l'ordonnement des attributs dans le message de réponse;
- spécifier la matrice de données réelle du message de réponse;
- combiner les deux méthodes ci-dessus afin que la matrice de données réelle puisse contenir également les numéros d'attributs.

Quelle que soit la méthode utilisée, les règles spécifiées dans le Tableau 1 doivent être respectées pour spécifier la Service_ResponsePDU d'une classe d'objets tant pour les attributs de classe que pour les attributs d'instance.

Tableau 1 – Règles relatives au service de réponse Get_Attribute_All

Numéro de règle	Règle
1	Si la définition de la réponse Get_Attribute_All inclut des attributs facultatifs, les valeurs par défaut doivent être spécifiées dans la description de la réponse. Ces valeurs par défaut doivent être utilisées si une mise en œuvre ne prend pas en charge les attributs facultatifs. Si l'un quelconque des attributs facultatifs suivants sont inclus dans une spécification d'objet, mais ne sont pas pris en charge dans la mise en œuvre de l'objet, les conditions suivantes doivent être satisfaites: – Si l'attribut de classe "Optional attribute list" (c'est-à-dire: Liste d'attributs facultatifs) n'est pas pris en charge, la valeur par défaut zéro doit être insérée dans la matrice de réponse et aucun numéro d'attribut facultatif ne doit suivre; – Si l'attribut de classe "optional service list" (c'est-à-dire: Liste de services facultatifs) n'est pas pris en charge, la valeur par défaut zéro doit être insérée dans la matrice de réponse et aucun numéro de service facultatif ne doit suivre;

3.5.1.2.2 Révisions

La réponse `Get_Attribute_All` définie pour un objet peut voir sa taille augmenter après chaque révision de l'objet; cependant, afin d'assurer l'interopérabilité, le format de la première partie de la réponse doit être analysable par un client de la révision plus ancienne de l'objet. Les Clients (Demandeurs) peuvent ne pas utiliser cette exigence de compatibilité.

NOTE L'attribut de classe `Revision` n'est pas la révision d'une mise en œuvre (qui est reflétée dans l'objet `Identity`, bits de statut `Minor/Major revision` («révision mineure/majeure»), mais la révision de la *class definition* (définition de classe).

3.5.1.2.3 Demande `Set_Attribute_All`

3.5.1.2.3.1 Définition générale

Lorsque le service commun `Set_Attribute_All` est inclus dans la liste des services communs pris en charge, le format de la demande **`Set_Attribute_All`** doit être inclus dans la spécification de l'objet. La séquence ou l'ordre des données fournies dans la partie "Service Data" (données de service) de la demande doit être spécifié(e). Il existe trois façons dont il est permis de spécifier la demande `Set_Attribute_All`:

- énumérer l'ordre des attributs dans le message de demande;
- spécifier la matrice de données réelle du message de demande;
- combiner les deux méthodes ci-dessus afin que la matrice de données réelle puisse contenir également les numéros d'attributs.

Quelle que soit la méthode utilisée, les règles spécifiées dans le Tableau 4 doivent être respectées pour spécifier la demande `Set_Attribute_All` d'un objet dans la spécification de l'objet tant pour les attributs de classe que pour les attributs d'instance.

Tableau 4 – Règles relatives au service de demande `Set_Attribute_All`

Numéro de règle	Règle
1	Un objet doit prendre en charge le service <code>Set_Attribute_All</code> seulement si tous les attributs réglables montrés dans la demande <code>Set_Attribute_All</code> sont mis en œuvre comme étant réglables.
2	Si de nouveaux attributs réglables sont ajoutés à un objet existant, les attributs en question doivent être ajoutés à la fin de la liste d'attributs de demande ou matrice de données.
3	Quelle que soit la méthode utilisée pour spécifier la réponse, cela doit être fait de façon non ambiguë, en incluant les règles pour traiter des champs de longueur variable et du bourrage.
4	La demande <code>Set_Attribute_All</code> ne doit inclure que les attributs ouverts. Elle ne doit pas inclure d'attributs spécifiques à un vendeur.

Le Tableau 5 est un exemple de la méthode d'ordonnement des attributs permettant de spécifier la partie "données de service" d'une demande `Set_Attribute_All` pour des attributs de niveau instance relatifs à un objet qui prend en charge les attributs d'instance réglables exigés 7, 8, 9, 10, 11 et 12.

Tableau 5 – Exemple de méthode d'ordonnement d'attributs `Set_Attribute_All`

ID d'attribut de classe	Nom d'attribut
7	Output Range
8	Value Data Type
9	Fault State
10	Idle State

ID d'attribut de classe	Nom d'attribut
11	Fault Value
12	Idle Value

Le Tableau 6 est un exemple de la méthode de matrice de données permettant de spécifier la partie "données de service" d'une demande Set_Attribute_All pour des attributs de niveau instance relatifs à un objet qui prend en charge les attributs d'instance réglables exigés 7, 8, 9, 10, 11 et 12.

Tableau 6 – Exemple de méthode de matrice de données Set_Attribute_All

Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	Output Range							
1	Value Data Type							
2	Fault State							
3	Idle State							
4	Fault Value (octet de poids faible)							
5	Fault Value (octet de poids fort)							
6	Idle Value (octet de poids faible)							
7	Idle Value (octet de poids fort)							

3.5.1.2.3.2 Révisions

Un serveur traitant une demande Set_Attribute_All peut avoir besoin de traiter plus de données qu'il n'en est prévu par le serveur si le client a mis en œuvre une plus récente révision de cet objet alors que le serveur a une version plus ancienne. Il en résulte que l'objet serveur peut avoir à traiter une demande Set_Attribute_All qui contient plus de données parce que les attributs supplémentaires n'avaient pas été reconnus. Si le serveur reçoit plus de données dans une demande Set_Attribute_All qu'il n'en attend, le serveur doit répondre par un code de statut général égal à *Too much data*, c'est-à-dire, "trop de données" (0x15).

NOTE L'attribut de classe Revision n'est pas la révision d'une mise en œuvre (qui est reflétée dans l'objet Identity, bits de statut Minor/Major revision («révision mineure/majeure»), mais la révision de la *class definition* (définition de classe).

3.6 Conventions utilisées dans les diagrammes d'états

3.6.1 Conventions dans les diagrammes d'état pour Type 2

3.6.1.1 Généralités

Les changements d'état peuvent être déclenchés par des événements (internes ou externes) ou par des invocations de services. La réaction à des invocations de services peuvent dépendre de la valeur/des valeurs de l'attribut/des attributs accessible(s) au service.

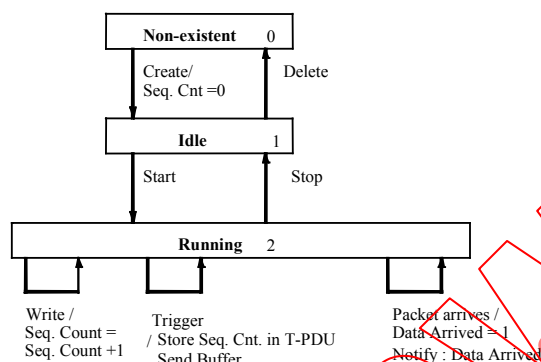
Le comportement du diagramme d'états doit être défini pour la combinaison de:

- l'événement que le diagramme reçoit;
- l'état dans lequel se trouve le diagramme lorsqu'il reçoit la notification d'un événement modificateur d'état.

Pour définir le comportement en ces termes, un State Transition Diagram (STD, Diagramme des transitions d'états) et une State Event Matrix (SEM, Matrice d'événements d'états) sont utilisés lorsque cela est applicable dans la spécification du diagramme d'états.

3.6.1.2 State Transition Diagram STD

Un **événement** est un stimulus externe qui peut entraîner une transition d'état. Un STD illustre graphiquement les états d'un objet et inclut des événements, des appels de services et des changements d'attributs qui le font passer à un autre état. La Figure 3 montre un exemple de STD.



Légende

Anglais	Français
Non-existent	Inexistant
Create/	Créer/
Seq. Cnt =0	Compte de séquence =0
Delete	Supprimer
Idle	Au repos
Start	Démarrer
Stop	Arrêter
Running	En marche
Write /	Écrire/
Seq. Count =	Compte de séquence =
Seq. Count +1	Compte de séquence +1
Trigger	Déclencher
/ Store Seq. Cnt. In T-PDU	/Stocker Seq. Cnt. dans la T-PDU
Send Buffer	Envoyer tampon
Packet arrives /	Paquet arrive/
Data Arrived = 1	Données arrivées = 1
Notify: Data Arrived	Notifier: Données arrivées

Figure 3 – Exemple d'un STD

NOTE L'événement qui est la cause principale de la transition d'état est suivi d'une barre "/". La notification à la couche supérieure est repérée par "Notify" (c'est-à-dire: notifier):

3.6.1.3 State Event Matrix (matrice d'évènement d'états)

Un **état** est le mode actuellement actif de fonctionnement de l'objet diagramme d'état (par exemple: Running (en marche), Idle (au repos)).

Une matrice d'événements d'états est un tableau qui énumère tous les possibles événements, services et modifications des valeurs d'attribut qui déclenchent un changement d'état. Elle indique également la réponse de l'objet à l'événement en fonction de l'état de l'objet lorsqu'il reçoit notification de l'événement en question.

Le format d'une matrice d'événements d'états est tel que montré dans le Tableau 7.

Tableau 7 – Format d'une matrice d'événements d'états

Événement	État		
	État 1		État n
Description de l'événement A	Fonction déclenchée par l'événement A dans l'État 1 (le cas échéant) Notification à l'utilisateur de FAL (le cas échéant) Transition vers un autre état (le cas échéant)		Fonction déclenchée par l'événement A dans l'État n (le cas échéant) Notification à l'utilisateur de FAL (le cas échéant) Transition vers un autre état (le cas échéant)
.....			
Description de l'Événement X	Fonction déclenchée par l'événement X dans l'État 1 (le cas échéant) Notification à l'utilisateur de FAL (le cas échéant) Transition vers un autre état (le cas échéant)		Fonction déclenchée par l'événement X dans l'État n (le cas échéant) Notification à l'utilisateur de FAL (le cas échéant) Transition vers un autre état (le cas échéant)
En l'absence de fonction, de notification ou de transition, l'entrée correspondante ne doit pas être faite dans le tableau.			

3.6.1.4 Exemple de matrice d'événements d'états

Le Tableau 8 montre un exemple d'un diagramme d'états avec trois états:

- *Non-existent* (inexistant): l'objet n'a pas encore été créé; les objets passent à l'état existant via le service "create" (si l'objet peut être créé de manière dynamique) ou à la mise sous tension (si l'objet est fixé par conception/mise en œuvre);
- *Idle (au repos)*: l'objet accepte des services (par exemple: Get_Attribute_Single), mais ne produit pas de données vers la liaison ou n'en consomme pas en provenance de la liaison;
- *Running (en marche)*: l'objet accomplit toutes ses fonctions spécifiées.

Tableau 8 – Exemple de matrice d'événements d'états

Événement	État		
	Non-existant (inexistant)	Idle (au repos)	Running (en marche)
Create	Passer à Idle	Erreur: L'objet existe déjà.	Erreur: L'objet existe déjà.
Delete	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Passer à Non-existant	Erreur: Object State Conflict, conflit d'états de l'objet (Code d'erreur général 0x0C)
Start	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Passer à Running	Erreur: Object State Conflict, conflit d'états de l'objet (Code d'erreur général 0x0C)
Stop	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Erreur: Object State Conflict, conflit d'états de l'objet (Code d'erreur général 0x0C)	Passer à Idle
Get_Attribute_Single	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Valider/gérer la demande. Retourner la réponse	Valider/gérer la demande. Retourner la réponse
Set_Attribute_Single	Erreur: L'objet n'existe pas. (Code d'erreur général 0x16)	Valider/gérer la demande. Retourner la réponse	Erreur: Object State Conflict, conflit d'états de l'objet (Code d'erreur général 0x0C)

4 Syntaxe abstraite

4.1 Syntaxe abstraite des PDU de la FAL

4.1.1 Généralités

La FAL_PDU doit être soit une UCMM_PDU, soit une Transport_PDU.

UCMM_PDU sert à acheminer de l'information pour les services sans connexion.

Transport_PDU sert à acheminer de l'information pour les services orientés connexion.

Un message connecté présume des ressources et des paramètres négociés au préalable à sa source, à sa/ses destination(s) et en tous les éventuels points de transit intermédiaires. Ces ressources sont référencées par un identificateur de connexion unique et peuvent ne pas être contenues dans chaque message. Seul l'ID de connexion est indispensable pour identifier le message et faire référence à ses paramètres connexes, permettant ainsi des économies considérables en termes de rendement de message.

Un message non connecté donne le moyen de communiquer sur la liaison locale sans ressources préalablement négociées en la destination et il doit donc transporter les détails complets de l'ID de la destination, les descripteurs de données internes et les détails complets de l'ID de la source si une réponse est demandée. Les messages non connectés sont surtout utilisés pour créer des connexions.

Le service non connecté est fourni par le gestionnaire de messages non connectés (UCMM «Unconnected Message Manager»). Les messages reçus par le biais de l'UCMM sont transmis au routeur de messages (MR «Message Router»), qui les dirige pour exécution vers

l'objet interne adéquat. Les connexions sont établies par des messages non connectés spécifiques envoyés par le biais du Connection Manager (CM, gestionnaire de connexion) à l'aide des services de l'UCMM. Les connexions peuvent être établies soit vers le Message Router (à des fins de messagerie), soit directement vers un objet d'application. La cible de la connexion est spécifiée à l'aide d'un chemin de connexion et peut être située sur une liaison locale ou distante, en passant par plusieurs nœuds routeurs intermédiaires. Une fois qu'une connexion est établie avec un objet d'application, l'UCMM, le MR et le CM ne sont plus indispensables, car les données seront échangées directement avec l'objet connecté, en fonction de l'ID de connexion correspondant. Les messages connectés envoyés au Message Router seront transmis pour exécution à l'objet interne adéquat.

4.1.2 Structure des PDU

4.1.2.1 Structure de UCMM_PDU

UCMM_PDU est une séquence d'un UCMM_Header, suivi soit de OM_Service (ASE Object Management), soit de CM_Service (ASE Connection Manager).

OM_Service doit être soit OM_Request, soit OM_Response.

OM_Request doit consister en MR_Request_Header et une Service_RequestPDU.

Service_RequestPDU doit être l'une des suivantes:

- Get_Attribute_Single_RequestPDU
- Get_Attribute_All_RequestPDU
- Get_Attribute_List_RequestPDU
- Set_Attribute_Single_RequestPDU
- Set_Attribute_All_RequestPDU
- Set_Attribute_List_RequestPDU
- Reset_RequestPDU
- Create_RequestPDU
- Delete_RequestPDU
- Start_RequestPDU
- Stop_RequestPDU
- Find_Next_Object_Instance_RequestPDU
- NOP_RequestPDU
- Apply_Attributes_RequestPDU
- Save_RequestPDU
- Restore_RequestPDU
- Group_Sync_RequestPDU
- Add_AckData_Path_RequestPDU
- Remove_AckData_Path_RequestPDU
- Get_Enum_String_RequestPDU
- Connection_Bind_RequestPDU
- Producing_Application_Lookup_RequestPDU

OM_Response doit consister en un MR_Response_Header et une Service_ResponsePDU.

Service_ResponsePDU doit être l'une des suivantes:

- Get_Attribute_Single_ResponsePDU
- Get_Attribute_All_ResponsePDU
- Get_Attribute_List_ResponsePDU
- Set_Attribute_Single_ResponsePDU
- Set_Attribute_All_ResponsePDU
- Set_Attribute_List_ResponsePDU
- Reset_ResponsePDU
- Create_ResponsePDU
- Delete_ResponsePDU
- Start_ResponsePDU
- Stop_ResponsePDU
- Find_Next_Object_Instance_ResponsePDU
- NOP_ResponsePDU
- Apply_Attributes_ResponsePDU
- Save_ResponsePDU
- Restore_ResponsePDU
- Group_Sync_ResponsePDU
- Add_AckData_Path_ResponsePDU
- Remove_AckData_Path_ResponsePDU
- Get_Enum_String_ResponsePDU
- Connection_Bind_ResponsePDU
- Producing_Application_Lookup_ResponsePDU

CM_Service doit être soit CM_Request, soit CM_Response.

CM_Request doit consister en MR_Request_Header et une CM_RequestPDU.

CM_RequestPDU doit être l'une des suivantes:

- Forward_Open_RequestPDU
- Forward_Close_RequestPDU
- Large_Forward_Open_RequestPDU
- Unconnected_Send_RequestPDU
- Get_Connection_Data_RequestPDU
- Search_Connection_Data_RequestPDU
- Get_Object_Owner_RequestPDU

CM_Response consiste en MR_Response_Header et un CM_ResponsePDU.

CM_ResponsePDU doit être l'une des suivantes:

- Forward_Open_ResponsePDU
- Forward_Close_ResponsePDU

- Large_Forward_Open_ResponsePDU
- Unconnected_Send_ResponsePDU
- Get_Connection_Data_ResponsePDU
- Search_Connection_Data_ResponsePDU
- Get_Object_Owner_ResponsePDU

4.1.2.2 Structure de Transport_PDU

Transport_PDU est une séquence d'un Transport_Header, suivi soit de OM_Service (ASE Object Management), soit de Application Data (données d'application).

OM_Service est défini en 4.1.2.1 ci-dessus.

Les données d'application proviennent de la source à laquelle la connexion a été faite.

4.1.3 UCMM_PDU

Toutes les UCMM_PDU doivent être envoyées et reçues en utilisant l'étiquette fixe 0x83 ou l'étiquette de gestion "Management tag" 0x88. Lorsqu'un dispositif est alimenté en énergie, l'UCMM doit invoquer le service `DLL_enable_fixed_request` de la couche liaison de données pour demander que les Lpackets à étiquette fixe 0x83 ou 0x88 soient acheminés vers l'UCMM. Tout l'envoi et toute la réception des TPDU doivent respectivement utiliser les primitives de service `DLL_fixed_request_req` et `DLL_fixed_request_ind` de la couche liaison de données. Le paramètre paquet de ces services doit être préfixé avec l'en-tête suivant pour former la PDU Transport avant l'envoi vers la couche liaison de données. Le format de l'en-tête de l'UCMM_PDU est montré dans le Tableau 9.

Tableau 9 – Format d'en-tête de l'UCMM_PDU

Nom de paramètre	Format
command_code	USINT
Timeout	USINT
TransactionID	UINT

Le paramètre `command_code` doit spécifier le type de l'UCMM_PDU comme montré dans le Tableau 10. Les paquets UCMM reçus avec un `command_code` réservé doivent être rejetés sans acquittement.

Tableau 10 – Codes de commande de l'UCMM

Command_code	Description
0	Reserved
1	acquitter une demande
2	demande avec répétition de tentative jusqu'à acquittement
3	réponse avec répétition de tentative jusqu'à acquittement
4	demande sans acquittement et sans réponse
5	acquitter une réponse
6	réponse qui ne doit pas répéter de tentative (pas d'acquittement)
7	demande avec répétition de tentative jusqu'à la réponse (pas d'acquittement)

Command_code	Description
8	Demande qui ne doit pas répéter de tentative et doit donner lieu à une réponse de code 6
9 à 255	Reserved

Le champ timeout (expiration de délai) doit spécifier la durée de la transaction dans un format de virgule flottante de 8 bits. Les 5 bits de poids fort du champ doivent être un exposant non signé qui n'est pas biaisé. Les 3 bits de poids faible doivent être les 3 bits de poids faible d'une mantisse non signée de 4 bits. Le bit de poids fort de la mantisse doit être 1 et ne doit pas apparaître de façon explicite dans la représentation en 8 bits. La virgule binaire doit être positionnée entre le 1 implicite et le reste de la mantisse. Les unités de la durée de transaction calculée avec l'aide du champ timeout doivent être des millisecondes.

NOTE En appliquant les règles de priorité de l'ANSI C, le nombre de tops d'horloge de 0,125 est $(8 \mid \text{timeout} \& 7) \ll (\text{timeout} \gg 3 \& 31)$.

Le champ transactionID doit être composé de deux sous-champs:

- les 10 bits de poids faible, RECORD, doivent identifier une transaction spécifique;
- les 6 bits de poids fort, SEQUENCE, doivent être utilisés pour la détection de doublons. Il doit être incrémenté pour chaque message unique figurant sur ce RECORD; cependant, il ne doit pas être incrémenté sur une répétition de tentative.

Un seul message doit être en cours pour un RECORD donné.

Si plusieurs messages en cours sont requis, plusieurs RECORD sont requis.

Lorsqu'un paquet à étiquette fixe "I'm alive" est reçu d'un nœud (voir la CEI 61158-4-2:2010, Station Management), toutes les transactions de l'UCMM avec le nœud en question doivent être abandonnées.

4.1.4 Transport_Header

4.1.4.1 Généralités

Le contenu des en-têtes Transport varie en fonction de la classe de transport sélectionnée au cours de l'établissement de la connexion.

4.1.4.2 Classe 0 (vide ou de base)

L'en-tête de transport de classe 0 doit être un nombre non signé de 16 bits. Cet en-tête doit être écrit dans le tampon de TPDU par le transport et doit être simplement rejeté, par le transport, du paquet entrant en provenance du consommateur. Le format de l'en-tête est montré dans le Tableau 11.

Tableau 11 – En-tête de transport de classe 0

Nom de paramètre	Format
don't_care	UINT

4.1.4.3 Classe 1 (détection de doublons)

L'en-tête de transport de classe 1 doit être un compte de séquences non signé de 16 bits. Cet en-tête doit être écrit dans le tampon de TPDU par le transport et doit être lu par le transport

dans le paquet entrant en provenance du consommateur. Le format de l'en-tête est montré dans le Tableau 12.

Tableau 12 – En-tête de transport de classe 1

Nom de paramètre	Format
sequence_count	UINT

4.1.4.4 Classe 2 (acquitté)

Comme l'en-tête de transport de classe 1, l'en-tête de transport de classe 2 doit être un compte de séquences de 16 bits. Cet en-tête doit être écrit dans le tampon de TPDU par le transport et doit être lu par le transport dans le paquet entrant en provenance du consommateur. Le format de l'en-tête est montré dans le Tableau 13.

Tableau 13 – En-tête de transport de classe 2

Nom de paramètre	Format
sequence_count	UINT

4.1.4.5 Classe 3 (vérifié)

Comme l'en-tête de transport de classe 1, l'en-tête de transport de classe 3 doit être un compte de séquences de 16 bits. Cet en-tête doit être écrit dans le tampon de TPDU par le transport et doit être lu par le transport dans le paquet entrant en provenance du consommateur. Le format de l'en-tête est montré dans le Tableau 14.

Tableau 14 – En-tête de transport de classe 3

Nom de paramètre	Format
sequence_count	UINT

4.1.4.6 Classe 4 (non bloquante), classe 5 (non bloquante, fragmentante) et classe 6 (connexion multipoint, fragmentante)

Ces classes de transports utilisent toutes le format d'en-tête montré dans le Tableau 15. Les classes 4 et 5 ont deux en-têtes dans chaque TPDU, un Sender (expéditeur) et un Receiver (récepteur). La Classe 6 utilise un en-tête Sender dans le sens Client vers Serveur et un en-tête Receiver dans le sens Serveur vers Client.

Tableau 15 – Format des en-têtes de classes 4 à 6

Command			Sequence Number (Numéro de séquence)	Data (Données)	
MSB (poids fort)	Bits 15 à 12		Bits 11 à 8	Bits 7 à 0	LSB
Valeur :	Signification dans l'en-tête Sender:	Signification dans l'en- tête Receiver:	Incréméte de 0 à 15, puis repasse à zéro. Utilisé dans l'en-tête Sender pour distinguer une répétition de tentative d'une nouvelle émission. Utilisé dans l'en-tête Receiver pour indiquer quelle TPDU d'envoi est ACK'd (acquittée) ou NAK'd (acquittée négativement).	Pour "First": Nombre de blocs de 256 octets (moins 1) à allouer pour contenir le message complet. Pour "NAK": Code de cause. Pour les autres: Mettre à zéro lors de la génération; ignorer lors de la réception.	
0.	Null	ACK			
1.	Only (c'est-à-dire: Seule)	NAK			
2.	First (c'est-à-dire: Première)	Not Initialized (c'est-à- dire: Non initialisée)			
3.	Middle (c'est-à-dire: Milieu)	Undefined (c'est-à-dire: Non définie)			
4.	Last (c'est-à-dire: Dernière)	Undefined (c'est-à-dire: Non définie)			
5. à 15 sont non définis.					

La Command identifie l'utilisation de la TPDU par le transport. Le champ Command est défini différemment pour l'en-tête Sender et pour l'en-tête Receiver.

Commandes de l'en-tête Sender.

Null = aucune donnée dans cette PDU Transport

Only = Il s'agit de la PDU Transport acheminant la totalité des données d'application.

First / Middle / Last = position de cette PDU Transport dans les données d'application segmentées

Le champ Sequence Number est utilisé dans l'en-tête Sender pour distinguer une répétition de tentative d'une nouvelle émission. Il est utilisé dans l'en-tête Receiver pour indiquer quelle TPDU d'envoi est ACK'd (acquittée) ou NAK'd (acquittée négativement). Il **n'est pas** utilisé pour la mise en correspondance de demande/réponse d'applications. Il est seulement utilisé par les diagrammes d'états de transport.

Le champ Data est utilisé par une commande de Sender et une commande de Receiver. Il est utilisé dans la commande de Sender *First* pour indiquer le nombre de blocs de 256 octets (moins 1) que le Receiver doit allouer pour le message d'application complet. À savoir, les messages d'application de taille de 0 octet à 256 octets doivent avoir le champ Data *First* mis à 0; de 257 à 512, mis à 1; et ainsi de suite jusqu'à 65 536 octets. Il est utilisé dans la commande de Receiver NAK pour fournir un code de cause:

La classe 4 (le transport non bloquant point à point) utilise deux en-têtes dans chaque TPDU, suivis des données d'application. L'en-tête Sender est le premier dans la TPDU, l'en-tête

Receiver est le second, suivi des données d'application. La classe 4 utilise seulement les commandes de Sender *Null* et *Only* (numérotées 0 et 1).

La classe 5 (le transport fragmentant point à point) utilise deux en-têtes dans chaque TPDU, suivis des données d'application. L'en-tête Sender est le premier dans la TPDU, l'en-tête Receiver est le second, suivi des données d'application. La classe 5 utilise toutes les Command de Sender.

La classe 6 (le transport fragmentant en connexion multipoint) utilise un en-tête dans chaque TPDU, suivi des données d'application. La classe 6 utilise toutes les Command de Sender. L'en-tête Sender est utilisé dans la TPDU Client vers Serveur et l'en-tête Receiver est utilisé dans la TPDU Serveur vers Client.

4.1.4.7 Exclusive owner O⇒T data format

Les connexions de propriétaire exclusif doivent avoir l'un des deux formats de transfert temps réel:

- en-tête de 32 bits, taille fixe;
- absence d'en-tête, taille variable.

L'en-tête de 32 bits préfixé aux données temps réel doit avoir la forme montrée dans le Tableau 16.

Tableau 16 – En-tête de données temps réel – propriétaire exclusif

Paramètre	Format	Longueur (bits)
Run_idle	RUN_IDLE	1
Reserved	UDINT	31

Le fanion `run_idle` (bit 0) doit être mis (1 = RUN, c'est-à-dire EN MARCHE) pour indiquer que les données suivantes doivent être envoyées à l'application-cible. Il doit être effacé (0 = IDLE, c'est-à-dire AU REPOS) pour indiquer que l'événement repos doit être envoyé à l'application-cible. Le champ `reserved` (bits 1-31) doit être réservé et mis à 0.

Si le format de transfert `no_header` est utilisé, la réception d'un paquet au-delà de l'en-tête de transport doit indiquer le mode RUN. Si le paquet est tronqué après l'en-tête de transport, la cible doit être un événement de repos.

4.1.4.8 Redundant owner O⇒T data format

4.1.4.8.1 Format général

Le propriétaire redondant doit avoir un en-tête de 32 bits préfixé aux données temps réel. Cet en-tête doit avoir la forme montrée dans le Tableau 17.

Tableau 17 – En-tête de données temps réel – propriétaire redondant

Paramètre	Format	Longueur (bits)
Run_idle	RUN_IDLE	1
COO	BOOLEAN	1
ROO	USINT	2
Reserved	UDINT	28

4.1.4.8.2 Fanion Run_idle

Le fanion run_idle (bit 0) doit avoir la même signification qu'il a dans la connexion de propriétaire exclusif et doit être soit RUN, soit IDLE. Le champ «reserved» (bits 4-31) doit être réservé et mis à 0.

4.1.4.8.3 Claim output ownership (COO) flag

Le fanion COO (revendiquer la propriété de la sortie) doit être mis (1) lorsqu'une application d'émetteur souhaite que sa connexion soit la connexion propriétaire de l'application-cible. Le fanion COO doit être réinitialisé (0) lorsqu'une application d'émetteur ne souhaite pas que sa connexion soit la connexion propriétaire de l'application-cible. Lorsque la connexion propriétaire réinitialise (0) son fanion COO, ses connexions enfants doivent être contrôlées pour détecter la présence d'un fanion COO mis (1). Le nouveau propriétaire doit être l'une quelconque des connexions dont le fanion COO est mis.

NOTE Il en résulte un comportement indéterminé si plusieurs connexions ont également leur fanion COO mis.

4.1.4.8.4 Valeur de priorité Ready for ownership of outputs (ROO)

La valeur de priorité ROO doit être non nulle lorsqu'une application d'émetteur ne souhaite pas forcer sa connexion à être la connexion propriétaire de l'application-cible, mais est prête à être la connexion propriétaire s'il n'y a pas d'application d'émetteur prétendant être la connexion propriétaire. La valeur de priorité ROO doit être zéro lorsque l'application d'émetteur ne souhaite pas être la connexion propriétaire de la connexion-cible, et n'est pas prévue être la connexion propriétaire s'il n'y a pas d'application d'émetteur prétendant être la connexion propriétaire. La valeur de priorité ROO doit être utilisée seulement lorsque le fanion COO est réinitialisé.

La valeur du champ ROO peut se situer dans la plage de 0 à 3. Les applications d'émetteur doivent déterminer chacune une unique valeur ROO non nulle.

4.1.4.8.5 Détermination de la connexion propriétaire

Les applications d'émetteur doivent déterminer laquelle parmi elles a la connexion propriétaire. La connexion propriétaire doit être déterminée par l'application d'émetteur qui met son fanion COO. Dans les situations dans lesquelles plusieurs applications d'émetteur ont leur fanion COO mis ou dans lesquelles aucune connexion d'émetteur n'a son fanion mis, les règles ci-après doivent être appliquées par le transport-cible pour déterminer la connexion propriétaire.

- Il ne doit y avoir aucune connexion propriétaire tant qu'une application d'émetteur n'a pas envoyé un paquet temps réel avec le fanion COO mis;
- S'il n'y a qu'une seule application d'émetteur qui a eu le fanion COO mis dans son dernier paquet temps réel, l'application d'émetteur en question doit avoir la connexion propriétaire;

- S'il y a plusieurs applications d'émetteur qui ont eu le fanion COO mis dans leur dernier paquet temps réel, la dernière application d'émetteur qui a fait passer son fanion COO de l'état réinitialisé à l'état mis doit avoir la connexion propriétaire.
- Si l'application d'émetteur avec la connexion propriétaire réinitialise son fanion COO, ferme sa connexion ou si la connexion en question expire, et aucune autre application d'émetteur n'a son fanion mis, l'application d'émetteur ayant la priorité ROO non nulle la plus élevée doit avoir la connexion propriétaire.
- Si toutes les applications d'émetteur ont leurs fanions COO réinitialisés et des valeurs de priorité ROO mises à zéro, il ne doit pas y avoir de connexion propriétaire.
- Lorsque le premier paquet temps réel contenant un fanion COO mis est reçu par le transport-cible, l'application d'émetteur qui a envoyé le paquet temps réel doit avoir la connexion propriétaire.

4.1.4.8.6 Transport d'événements et de données vers une application-cible

Les événements liés à la connexion issus de chacune des connexions de propriétaires redondants doivent être combinés à l'aide des règles ci-après de manière que l'application-cible ne voie qu'une seule connexion de propriétaire exclusif:

- Tant qu'une connexion propriétaire n'a pas été déterminée initialement, le transport ne doit pas indiquer de réception de données temps réel à l'application-cible;
- Si une connexion propriétaire est déterminée, le transport doit indiquer à l'application-cible l'événement compatible avec les données temps réel des connexions propriétaires. Si le fanion Run/Idle est réinitialisé dans les données temps réel pour la connexion propriétaire, le transport doit indiquer l'état de repos Idle à l'application-cible. Si le fanion Run/Idle est mis dans les données temps réel pour la connexion propriétaire, le transport doit indiquer l'état de marche RUN et les données temps réel à l'application-cible.
- Si une connexion propriétaire a été déterminée au préalable, mais aucun émetteur n'en réclame la propriété ni n'est prêt pour la propriété, le transport doit indiquer l'état de repos Idle à l'application-cible;
- Si toutes les connexions redondantes sont fermées ou ont été interrompues par temporisation, le transport-cible doit indiquer à l'application-cible l'événement compatible avec la dernière connexion à avoir été fermée ou interrompue par temporisation.

4.1.5 CM_PDU

4.1.5.1 Généralités

Les services d'établissement et de maintenance de connexion sont fournis par le gestionnaire de connexion Connection Manager. Ce sont

CM_Forward_Open (correspond aux PDU Forward_Open et Ex_Forward_Open)
 CM_Forward_Close (correspond aux PDU Forward_Close)
 CM_Unconnected_Send (correspond aux PDU Unconnected_Send)
 CM_Get_Connection_Data (correspond aux PDU Get_Connection_Data)
 CM_Search_Connection_Data (correspond aux PDU Search_Connection_Data)
 CM_Get_Object_Owner (correspond aux PDU Get_Object_Owner)

4.1.5.2 Connection Manager

L'objet Connection Manager sert à gérer l'établissement et la maintenance des connexions de communication.

Les objets Connection Manager situés aux différents nœuds doivent communiquer à l'aide des services de l'UCMM décrits dans la CEI 61158-5-2:2010. Les TPDU avec lesquelles communiquent les Connection Manager d'homologues doivent être dérivées des services du Connection Manager.

4.1.5.3 Forward_Open

4.1.5.3.1 Généralités

La réponse à la Forward_Open_Request_PDU est une Forward_Open_ResponsePDU. La réponse Forward_Open doit contenir le numéro de série de connexion originale, le numéro de série et de vendeur du propriétaire et les ID de connexion (CID) nécessaires pour parachever la connexion si succès et des informations d'erreur si échec de la connexion.

NOTE 1 La demande Forward_Open établit les connexions de réseau, de transport et d'application. Une connexion d'application consiste en une seule connexion de transport et une ou deux connexions de réseau qui sont constituées elles-mêmes de plusieurs connexions de liaison. Chaque segment de port dans le chemin de connexion utilise une connexion de liaison. Le service Forward_Open entre deux dispositifs construit une ou deux connexions de liaison telles que spécifiées par le paramètre de connexion de réseau et les intervalles requis entre paquets (CM_RPI). Sachant qu'il peut être requis jusqu'à deux connexions réseau pour une seule connexion de transport, celles-ci sont différenciées par les désignations O $\Rightarrow$ T et T $\Rightarrow$ O; O $\Rightarrow$ T signifie "originator to target" (c'est-à-dire: émetteur vers cible) et T $\Rightarrow$ O signifie "target to originator" (c'est-à-dire: cible vers émetteur).

NOTE 2 Le traitement d'un seul service Forward Open peut donner lieu à la création de plusieurs instances actives de l'objet Connection.

4.1.5.3.2 Format de la demande Forward_Open

Le format de la TPDU de demande Forward_Open doit avoir la forme montrée dans le Tableau 18.

Tableau 18 – Format de la demande Forward_Open

Nom de paramètre	Format
priority_and_tick	SWORD
connection time-out ticks	USINT
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_connection_parameters	UINT
T2O_CM_RPI	UDINT
T2O_connection_parameters	UINT
xport_type_and_trigger	SWORD
connection_path_size	USINT
connection_path	Padded EPATH

4.1.5.3.3 Format de la réponse Forward_Open si succès

Si le paramètre `status` de la `CM_open_response` est zéro (absence d'erreur), le format de la TPDU de réponse Forward_Open doit avoir la forme montrée dans le Tableau 19.

Tableau 19 – Format de la réponse Forward_Open_Good

Nom de paramètre	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
O2T_CM_API	UDINT
T2O_CM_API	UDINT
Application_reply_size; (en mots de 16 bits)	USINT
Reserved	USINT
application_reply[]	UINT

Success (succès) doit être retourné lorsque la connexion demandée a été établie à partir de point dans le chemin. Cette réponse doit aussi indiquer le numéro de série de la connexion et le débit de paquets réel de la connexion. Une fois qu'une réponse réussie a été reçue, la connexion doit être ouverte à partir de point dans le chemin. Les cibles doivent attendre au moins 10 secondes après l'envoi de CM_Forward_Open_Good_Response pour le premier paquet sur une connexion.

4.1.5.3.4 Format de la réponse Forward_Open si échec

Si le paramètre status de la CM_open_response est non nul (erreur), le format de la TPDU de réponse Forward_Open doit avoir la forme montrée dans le Tableau 20.

Tableau 20 – Format de la réponse Forward_Open_Bad

Nom de paramètre	Format
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size; (en mots de 16 bits)	USINT
Reserved	USINT

Ce format doit être utilisé pour toutes les défaillances du système de connexions. La connexion demandée ne doit pas être établie et les mots de statut spécifiques à l'objet doivent contenir des informations relatives à la cause de la défaillance. Le paramètre remaining_path_size doit contenir la longueur du chemin au point où la connexion a failli. Cette information peut être utilisée pour déboguer le problème.

Dans la réponse d'échec, le paramètre restant remaining_path_size doit avoir une longueur "pre-stripped" («précouchée»). Il doit s'agir de la longueur du chemin lorsque le nœud reçoit d'abord la demande et n'a pas encore commencé à la traiter. Un nœud-cible peut retourner soit la longueur «précouchée», soit 0 pour le remaining_path_size restant.

Un doublon de service Forward_Open doit être défini comme un service Forward_Open dont les vendor_ID, connection_serial_number et originator_serial_number correspondent aux paramètres d'une connexion existante. Si le doublon de service Forward_Open est un service Forward_Open vide (défini comme le type de connexion dans

lequel les champs de paramètres de connexion réseau tant O2T que T2O sont NULL), le service Forward_Open doit être transmis à l'application pour un traitement ultérieur. Les demandes Forward_Open vides ("null") peuvent être utilisées pour reconfigurer la connexion. Le Connection Manager dans les nœuds intermédiaires peut ne pas allouer de ressources supplémentaires pour un doublon de demande Forward_Open, car les ressources ont déjà été allouées. Si le doublon de demande Forward_Open n'est pas NULL, un statut général = 0x01, statut étendu = 0x0100, doit être retourné.

Il convient que le comportement d'émetteur dans le cas du doublon de Forward_Open soit de fermer et rétablir la connexion ou bien d'attendre l'expiration de la connexion et la rétablir. Il doit être reconnu que dans le second cas, il doit prendre 60 secondes (première expiration de données) à la connexion pour expirer avant qu'elle ne puisse être rétablie.

4.1.5.4 Large_Forward_Open

4.1.5.4.1 Généralités

Large_Forward_Open doit avoir la même fonction que Forward_Open, excepté qu'il doit permettre l'établissement de connexions supérieures à 511 octets.

La réponse à la Large_Forward_Open_Request_RDU est une Large_Forward_Open_ResponsePDU. La réponse Large_Forward_Open doit contenir le numéro de série de connexion originale, le numéro de série et de vendeur du propriétaire et les ID de connexion (CID) nécessaires pour parachever la connexion si succès et des informations d'erreur si échec de la connexion.

NOTE La demande Large_Forward_Open est très similaire à la demande Forward_Open.

4.1.5.4.2 Format de la demande Large_Forward_Open

Le format de la TPDU de demande Large_Forward_Open doit avoir la forme montrée dans le Tableau 21.

Tableau 21 – Format de la demande Large_Forward_Open

Nom de paramètre	Format
priority_and_tick	SWORD
connection time-out ticks	USINT
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_ex_connection_size	UINT
O2T_connection_parameters	UDINT
T2O_CM_RPI	UDINT
T2O_ex_connection_size	UINT
T2O_connection_parameters	UDINT
xport_type_and_trigger	SWORD
connection_path_size	USINT
connection_path	Padded EPATH

Le champ longueur de la connexion dans les deux connection_parameters doit être réservé dans ce cas et mis à zéro. Les paramètres ex_connection_size doivent être utilisés à la place pour spécifier la longueur maximale de la connexion (jusqu'à 65 535).

4.1.5.4.3 Format de la réponse Large_Forward_Open si succès

Si le paramètre status de la CM_open_response est zéro (absence d'erreur), le format de la TPDU de réponse Large_Forward_Open doit avoir la forme montrée dans le Tableau 22.

Tableau 22 – Format de la réponse Large_Forward_Open_Good

Nom de paramètre	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
O2T_CM_API	UDINT
T2O_CM_API	UDINT
Application_reply_size; (en mots de 16 bits)	USINT
Reserved	USINT
application_reply[]	UINT

Success (succès) doit être retourné lorsque la connexion demandée a été établie à partir de point dans le chemin. Cette réponse doit aussi indiquer le numéro de série de la connexion et le débit de paquets réel de la connexion. Une fois qu'une réponse réussie a été reçue, la connexion doit être ouverte à partir de point dans le chemin.

4.1.5.4.4 Format de la réponse Large_Forward_Open si échec

Si le paramètre `status` de la `CM_open_response` est non nul (erreur), le format de la TPDU de réponse `Large_Forward_Open` doit avoir la forme montrée dans le Tableau 23.

Tableau 23 – Format de la réponse Large_Forward_Open_Bad

Nom de paramètre	Format
<code>connection_serial_number</code>	UINT
<code>vendor_ID</code>	UINT
<code>originator_serial_number</code>	UDINT
<code>remaining_path_size</code> ; (en mots de 16 bits)	USINT
Reserved	USINT

Soit un dispositif-cible, soit un routeur intermédiaire doit retourner un code d'erreur étendu «Invalid Connection Size», avec le mot de statut supplémentaire de la longueur maximale de connexion prise en charge, si la longueur de la connexion demandée n'est pas prise en charge lors du traitement de la demande `Large_Forward_Open`.

Ce format doit être utilisé pour toutes les défaillances du système de connexions. La connexion demandée ne doit pas être établie et les mots de statut spécifiques à l'objet doivent contenir des informations relatives à la cause de la défaillance. Le paramètre `remaining_path_size` doit contenir la longueur du chemin au point où la connexion a failli. Cette information peut être utilisée pour déboguer le problème.

Dans la réponse d'échec, le paramètre restant `remaining_path_size` doit avoir une longueur "pre-stripped" («précouchée»). Il doit s'agir de la longueur du chemin lorsque le nœud reçoit d'abord la demande et n'a pas encore commencé à la traiter. Un nœud-cible peut retourner soit la longueur «précouchée», soit 0 pour le `remaining_path_size` restant.

Il convient que le comportement d'émetteur dans le cas du doublon de `Forward_Open` soit de fermer et rétablir la connexion ou bien d'attendre l'expiration de la connexion et la rétablir. Il doit être reconnu que dans le second cas, il doit prendre 60 secondes (première expiration de données) à la connexion pour expirer avant qu'elle ne puisse être rétablie.

4.1.5.5 Forward_Close

4.1.5.5.1 Généralités

La demande `Forward_Close` doit retirer une connexion de tous les nœuds participant à la connexion d'origine. La demande `Forward_Close` doit être envoyée entre les Connection Manager comme spécifié dans le `connection_path`. La demande `Forward_Close` doit entraîner la désaffectation de toutes les ressources dans tous les nœuds participant à la connexion, y compris les ID de connexion, le temps d'émission de liaison et les tampons des mémoires internes.

Si un nœud intermédiaire ne peut pas trouver la connexion qui est à fermer (elle peut avoir expiré au nœud), la demande `Forward_Close` doit encore être transmise aux nœuds aval ou à l'application-cible (par le truchement de `CM_close_indication`).

NOTE Forward_Close est toujours transmis pour permettre à des nœuds aval ou à l'application-cible de libérer les éventuelles ressources qui étaient allouées pour la connexion.

4.1.5.5.2 Format de Forward_Close request

Le format de demande Forward_Close doit avoir la forme montrée dans le Tableau 24.

Tableau 24 – Format de la demande Forward_Close

Nom de paramètre	Format
connection_priority/tick time	SWORD
connection_timeout	USINT
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
connection_path_size	USINT
Reserved	USINT
connection_path	Padded EPATH

4.1.5.5.3 Format de la réponse Forward_Close si succès

La réponse Forward_Close doit être envoyée entre les Connection Manager en réponse à une demande Forward_Close et doit contenir le statut de la demande de fermeture. Si le paramètre `status` de la `CM_close_response` est zéro (absence d'erreur), le format de la TPDU de réponse Forward_Close doit avoir la forme montrée dans le Tableau 25.

Tableau 25 – Format de la réponse Forward_Close_Good

Nom de paramètre	Format
connection serial number	UINT
vendor ID	UINT
originator serial number	UDINT
application_reply_size, en mots de 16 bits	USINT
Reserved	USINT
application_reply[]	UINT

Une réponse Forward_Close réussie doit être retournée lorsque la fermeture a été acquittée par tous les nœuds à partir de ce point du chemin jusqu'à la fin du chemin. La réponse doit indiquer les `connection_serial_number`, `vendor_ID` et `originator_serial_number` de la connexion fermée. Une fois qu'une réponse indiquant le succès a été reçue, la connexion doit être fermée à partir de ce point du chemin jusqu'à la fin. L'application-cible peut, facultativement, renvoyer des informations spécifiques à l'application dans la réponse de fermeture réussie. Si aucune information `application_reply` n'est contenue dans le service, le paramètre `application_reply_size` doit être zéro.

4.1.5.5.4 Format de la réponse Forward_Close si échec

Si le paramètre `status` de la `CM_close_response` est non nul (erreur), le format de la TPDU de réponse Forward_Close doit avoir la forme montrée dans le Tableau 26.

Tableau 26 – Format de la réponse Forward_Close_Bad

Nom de paramètre	Format
connection_serial_number	UINT
vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size	USINT
reserved (réservé)	USINT

Les mots de statut spécifiques à l'objet doivent contenir les informations relatives à la cause de l'échec. Le paramètre `remaining_path_size` doit contenir la longueur du chemin à partir du point auquel la connexion de fermeture a failli.

Dans la réponse d'échec, le paramètre `remaining_path_size` doit avoir une longueur "pre-stripped" («précouchée»). Il doit s'agir de la longueur du chemin lorsque le nœud reçoit d'abord la demande et n'a pas encore commencé à la traiter. Un nœud-cible doit retourner soit la longueur «précouchée», soit 0 pour le `remaining_path_size`.

4.1.5.6 Unconnected_Send

4.1.5.6.1 Format de la demande Unconnected_Send

Le service `Unconnected_Send` doit permettre à une application d'envoyer un message à un dispositif sans établir une connexion d'abord. Le service `Unconnected_Send` doit utiliser l'objet `Connection Manager` en chaque nœud intermédiaire pour transmettre le message et se rappeler le chemin de retour. L'UCMM de chaque liaison doit être utilisé pour transmettre la demande d'un `Connection Manager` à un autre tout comme dans le cas du service `Forward_Open`; toutefois, aucune connexion ne doit être bâtie. Le service `Unconnected_Send` doit être envoyé au `Connection Manager` local et doit être envoyé entre les nœuds intermédiaires. Lorsqu'un nœud intermédiaire enlève le dernier segment de port, le message doit être formaté comme un message d'UCMM et envoyé vers le port et l'adresse de liaison du dernier segment.

NOTE le nœud-cible ne voit jamais le service `Unconnected_Send`, mais voit seulement un message standard arrivant en passant par l'UCMM.

Si le message contient un nombre impair d'octets, un octet de bourrage doit être accolé à la fin du message permettant aux champs restants de s'aligner sur une frontière de 16 bits. Le format de la demande `Unconnected_Send` doit avoir la forme montrée dans le Tableau 27.

Tableau 27 – Format de demande Unconnected_Send

Nom de paramètre	Format
priority/tick time	SWORD
conn time-out ticks	USINT
Message_size; en octets, non compris le bourrage	UINT
Message Router PDU; si le paramètre <code>message_router_PDU</code> est de longueur impaire, inclure l'octet de bourrage.	MR_Request
pad	USINT
path_size	USINT
Reserved	USINT
path	Padded EPATH

Le paramètre `message_size` doit être la longueur du paramètre `message_router_PDU` en octets sans le `pad`. Le champ `reserved` doit être mis à zéro. Le paramètre `path_size` doit être le nombre de mots de 16 bits contenus dans le champ `path`.

4.1.5.6.2 Réponse `Unconnected_Send`

La réponse `Unconnected_Send` doit être générée par le dernier nœud intermédiaire à partir de la réponse de l'UCMM générée par le nœud-cible ou par un nœud intermédiaire à la suite d'une expiration d'UCMM, d'un problème avec le message incorporé ou d'un problème avec la demande de service non connectée ('Unconnected Service Request') elle-même. Le paquet doit être acheminé d'un nœud intermédiaire à l'autre en utilisant les informations stockées lorsque la demande `Unconnected_Send` était traitée. La réponse doit contenir un en-tête avec des informations de statut relatives à la demande et une réponse de longueur variable générée par le nœud-cible.

4.1.5.6.3 Format de la réponse `Unconnected_Send` si succès

Ce format doit être utilisé lorsqu'une réponse de message non connecté est reçue. La réponse d'application doit avoir le format de la réponse de message issue de la cible et doit contenir des codes d'erreur qui ont résulté de l'exécution du message par l'application au nœud-cible. La structure de la réponse est montrée dans le Tableau 28.

Tableau 28 – Format de la réponse `Unconnected_Send_Good`

Nom de paramètre	Format
<code>application_reply</code> []	OCTET

4.1.5.6.4 Format de la réponse `Unconnected_Send` si échec

Ce format doit être utilisé si le nœud émetteur ou n'importe quel nœud intermédiaire a connu un problème lié au format de message, aux paramètres de message ou un UCMM a temporisé en raison de la non-réception d'un acquittement ou d'une réponse. Une temporisation d'UCMM peut être le résultat

- du fait que le dispositif-cible ne soit pas connecté ou alimenté en énergie. (temporisation spécifique à une liaison);
- du fait qu'un dispositif intermédiaire ne soit pas connecté ou alimenté en énergie. (temporisation spécifique à une liaison);
- du fait qu'un dispositif-cible soit à court de tampons d'UCMM ou surchargé (Régulation de débit). (temporisation spécifique à une liaison);
- du fait qu'un dispositif intermédiaire soit à court de tampons d'UCMM ou surchargé (Régulation de débit). (temporisation spécifique à une liaison);
- d'un type de segment de chemin non valide (segment logique ou de réseau);
- d'une adresse de segment de port non valide (adresse de liaison ou de port non valide).

Comme le système de message non connecté ne met en œuvre aucune régulation de débit, la défaillance d'un message `Unconnected_Send` peut ne pas signifier qu'un dispositif soit absent du système. Le format du service de réponse `Unconnected_Send` pour une défaillance doit avoir la forme montrée dans le Tableau 29.

Tableau 29 – Format de la réponse Unconnected_Send_Bad

Nom de paramètre	Format
remaining_path_size	USINT
pad	USINT

Dans la réponse d'échec, le paramètre restant Connection Path Size doit avoir une longueur "pre-stripped" («précouchée»). Il doit s'agir de la longueur du chemin lorsque le nœud reçoit d'abord la demande et n'a pas encore commencé à la traiter. Un nœud-cible peut retourner soit la longueur «précouchée», soit 0 pour le Connection Path Size restant.

4.1.5.7 Get_Connection_Data

4.1.5.7.1 Généralités

Ce service doit retourner les paramètres associés à un numéro de connexion spécifié. Le connection number peut être différent d'un dispositif à l'autre même pour la même connexion. Le connection number correspond au décalage dans l'attribut de Connection Manager qui énumère le statut des connexions.

NOTE Ce service peut être utilisé pour les diagnostics de réseaux.

4.1.5.7.2 Format de la demande Get_Connection_Data

Le format de la demande Get_Connection_Data doit avoir la forme montrée dans le Tableau 30.

Tableau 30 – Format de la demande Get_Connection_Data

Nom de paramètre	Format
connection_number	UINT

4.1.5.7.3 Format de la réponse Get_Connection_Data

Le format de la TPDU de réponse Get_Connection_Data doit avoir la forme montrée dans le Tableau 31.

Tableau 31 – Format de la réponse Get_Connection_Data

Nom de paramètre	Format
connection_number	UINT
connection_state	UINT
originator_port	UINT
target_port	UINT
connection_serial_number	UINT
vendor_ID	UINT
serial_number	UDINT
originator_O2T_CID	UDINT
target_O2T_CID	UDINT
O2T_connection timeout multiplicier	USINT
reserved1[3]	USINT
originator_O2T_CM_RPI	UDINT
originator_O2T_CM_API	UDINT
originator_T2O_CID	UDINT
target_T2O_CID	UDINT
T2O_connection timeout multiplicier	USINT
reserved2[3]	USINT
originator_T2O_CM_RPI	UDINT
originator_T2O_CM_API	UDINT

4.1.5.8 Search_Connection_Data

4.1.5.8.1 Généralités

Ce service doit retourner les paramètres associés à une connexion spécifiée au sein d'un dispositif identifié par vendeur et numéro de série.

NOTE Ce service peut être utilisé pour les diagnostics de réseaux.

4.1.5.8.2 Format de la demande Search_Connection_Data

Le format de la demande Search_Connection_Data doit avoir la forme montrée dans le Tableau 32.

Tableau 32 – Format de la demande Search_Connection_Data

Nom de paramètre	Format
connection_serial_number	UINT
vendor_ID	UINT
serial_number	UDINT

4.1.5.8.3 Format de la réponse Search_Connection_Data

Le format de la réponse Search_Connection_Data doit être le même que celui de la réponse issue du service Get_Connection_Data (voir 4.1.5.7).

4.1.5.9 Get_Object_Owner

4.1.5.9.1 Généralités

Le service Get_Object_Owner du Connection Manager doit retourner des données concernant la/les connexion(s) qui possède(nt) un objet particulier. Il doit être mis en œuvre dans tout dispositif qui accepte des connexions redondantes.

4.1.5.9.2 Format de la demande Get_Object_Owner

Le format de la TPDU de demande Get_Object_Owner doit avoir la forme montrée dans le Tableau 33.

Tableau 33 – Format de la demande Get_Object_Owner

Nom de paramètre	Format
reserved (réservé)	USINT
path_size	USINT
path[]	UINT

Le champ "reserved" doit être mis à zéro.

4.1.5.9.3 Format de la réponse Get_Object_Owner

Le format de la TPDU de réponse Get_Object_Owner doit avoir la forme montrée dans le Tableau 34.

Tableau 34 – Format de la réponse Forward_Open_Good

Nom de paramètre	Format
number_of_connections	USINT
number_claiming_ownership	USINT
number_ready_for_ownership	USINT
last_action	USINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT

4.1.6 Composants de CM PDU

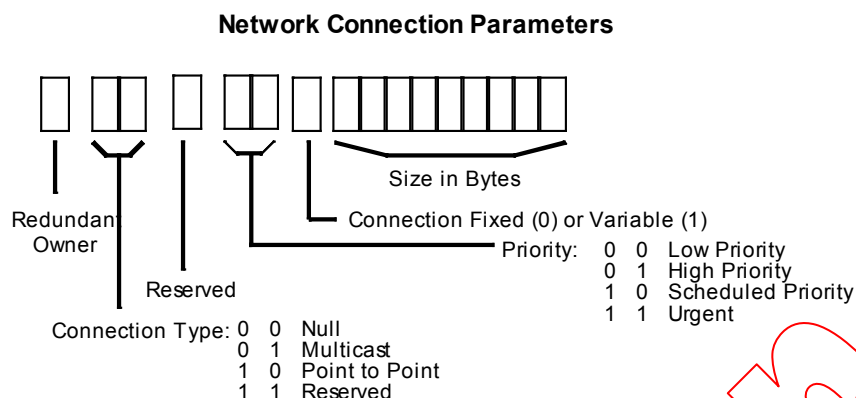
4.1.6.1 Paramètres de connexion réseau

4.1.6.1.1 Format

Les paramètres de connexion réseau doivent être fournis sous la forme d'un seul mot de 16 bits qui contient cinq champs, comme montré à la Figure 4. Les champs au sein du mot de 16 bits doivent indiquer

- le type de connexion réseau souhaitée;
- si la taille de tampon est variable ou fixe;
- la priorité de la connexion;
- la taille requise du tampon de connexion.

La taille du tampon de connexion doit inclure tout en-tête de transport.



Légende

Anglais	Français
Network Connection Parameters	Paramètres de connexion réseau
Redundant Owner	Propriétaire redondant
Size in Bytes	Taille en octets
Connection Fixed (0) or Variable (1)	Connexion Fixe (0) ou Variable (1)
Priority:	Priorité
0 0 Low Priority	Faible Priorité
0 1 High Priority	Haute Priorité
1 0 Scheduled Priority	Priorité programmée
1 1 Urgent	Urgent
Reserved	Réservé
Connection Type:	Type de connexion
0 0 Null	Vide
0 1 Multicast	Multidiffusion
1 0 Point to Point	Point à Point
1 1 Reserved	Réservés

Figure 4 – Paramètres de connexion réseau

4.1.6.1.2 Connection Type (type de connexion)

Le type de connexion doit avoir l'une des valeurs NULL, MULTIPOINT ou POINT2POINT. Le type NULL doit indiquer qu'aucune connexion réseau n'est indispensable, alors que MULTIPOINT et POINT2POINT renvoient aux types de connexion réseau tels que définis dans la CEI 61158-5-2:2010. Le type de connexion NULL peut être utilisé pour reconfigurer une connexion.

4.1.6.1.3 Priority (priorité)

Priorité doit être l'une des valeurs LOW, HIGH et SCHEDULED telles que spécifiées dans la CEI 61158-4-2:2010.

NOTE Les données SCHEDULED sont classées par ordre de priorité sur toute la liaison; cependant, les deux priorités non programmées (LOW et HIGH) sont arbitrées de façon indépendante en chaque nœud. Un nœud peut émettre un paquet de priorité LOW alors qu'un autre nœud a des données de priorité HIGH à envoyer.

4.1.6.1.4 Connection fixed/variable (connexion fixe/variable)

La connexion peut être établie pour utiliser des connexions de taille fixe ou variable. Avec une connexion à taille fixe, chaque émission sur la connexion doit utiliser le tampon de même taille. Autrement, il doit se produire une condition soit de dépassement de capacité de tampon, soit de sous-utilisation de capacité de tampon. Avec une connexion de taille variable, chaque émission sur la connexion peut envoyer une quantité variable de données jusqu'à la taille maximale, qui doit être spécifiée lorsque la connexion est ouverte. Les sous-utilisations de capacité de tampon ne doivent pas être relatées, mais un dépassement de capacité de tampon peut toujours se produire si trop de données sont envoyées.

4.1.6.1.5 Taille de connexion

La taille de la connexion réseau doit être la taille du tampon requise pour la connexion, qui inclut les données et tout en-tête de transport. Pour une connexion de taille variable, la taille doit être la taille maximale du tampon pour un transfert quelconque. La taille réelle du transfert pour une connexion variable doit être inférieure ou égale à la taille spécifiée pour la connexion réseau. La taille de tampon maximale doit être fonction des liaisons que la connexion traverse.

4.1.6.1.6 Redundant owner (propriétaire redondant)

Le bit "redundant owner" dans le sens O⇒T doit être mis (= 1) pour indiquer que plusieurs propriétaires peuvent être autorisés à établir simultanément une connexion. Le bit doit être effacé (= 0) pour indiquer une connexion en propriétaire exclusif, en saisie seule ou en écoute seule.

4.1.6.1.7 Paramètres réservés

Les bits Reserved (réservés) doivent être effacés (= 0).

4.1.6.2 Intervalle entre paquets

4.1.6.2.1 Intervalle demandé entre paquets (CM_RPI)

L'intervalle demandé entre les paquets doit être le temps demandé entre les paquets en microsecondes. Le format du CM_RPI doit être un nombre entier de 32 bits en microsecondes.

4.1.6.2.2 Intervalle réel entre paquets (CM_API)

L'intervalle réel entre paquets doit être le temps réel entre les paquets en microsecondes. Le format du CM_API doit être un nombre entier de 32 bits en microsecondes.

4.1.6.2.3 Usage

L'intervalle demandé entre les paquets doit être le temps entre paquets demandé par le dispositif de réception. La valeur doit être utilisée pour allouer un temps d'émission de liaison en chacun des nœuds producteurs. L'allocation du temps d'émission de liaison peut devoir être réajustée lorsque le débit réel de paquets ou l'intervalle réel entre paquets est retourné, car il est possible que les deux valeurs diffèrent. La valeur de temporisation de chemin en chacun des nœuds intermédiaires et cibles doit aussi être mise à la valeur du Connection time-out multiplier (multiplicateur de temporisation de connexion) multiplié par le CM_API. Le CM_RPI est donc requis pour toutes les connexions.

4.1.6.2.4 Fonction de priorités

Scheduled Priority

Pour la priorité programmée (Scheduled Priority), le CM_RPI doit être le débit de paquets des données répétitives. Sur des liaisons qui prennent en charge l'allocation de temps d'émission de liaison, le temps d'émission de liaison doit être réservé pour ce paquet. Pour la priorité programmée, les données doivent aussi être limitées au débit de paquets spécifié, ce qui signifie que si des données arrivent à un nœud intermédiaire plus rapidement que le débit de paquets spécifié, le nœud doit filtrer les paquets selon le débit spécifié. Comme la fréquence de mise à jour de la priorité programmée de chaque nœud est exprimée en quanta discrets, l'intervalle réel entre paquets (CM_API) peut être plus court (plus rapide) que le CM_RPI. La valeur de temporisation de chemin doit être mise à la valeur du Connection Time-out multiplier multiplié par le CM_API.

High Priority

Pour la haute priorité (High Priority), le CM_RPI doit être utilisé pour fixer la temporisation de chemin dans les nœuds intermédiaires et cibles. Le CM_RPI doit donc être réglé au plus lent débit de paquets, ce qui doit exclure de voir la connexion se fermer en raison d'une temporisation de chemin. Plus la temporisation de chemin est longue, plus long est le temps requis pour récupérer les ressources dans les nœuds intermédiaires en raison de défauts dans le réseau. La haute priorité n'étant quantifiée en aucun des nœuds, le CM_API doit être égal au CM_RPI. Toutefois, pour maintenir la cohérence, la valeur de temporisation doit de nouveau être mise à la valeur du Connection Time-out multiplier multiplié par le CM_API.

Low Priority

Pour la faible priorité (Low Priority), le CM_RPI doit être utilisé pour fixer la temporisation de chemin dans les nœuds intermédiaires et cibles. Le CM_RPI doit donc être réglé au plus lent débit de paquets, ce qui doit exclure de voir la connexion se fermer en raison d'une temporisation de chemin. Plus la temporisation de chemin est longue, plus long est le temps requis pour récupérer les ressources dans les nœuds intermédiaires en raison de défauts dans le réseau. La faible priorité n'étant quantifiée en aucun des nœuds, le CM_API doit être égal au CM_RPI. Toutefois, pour maintenir la cohérence, la valeur de temporisation doit de nouveau être mise à la valeur du Connection Time-out multiplier multiplié par le CM_API.

4.1.6.3 Connection time-out multiplier

Le Connection Time-out Multiplier (Multiplicateur de temporisation de connexion) spécifié le multiplicateur appliqué au CM_RPI pour obtenir la valeur de temporisation de connexion. Le dispositif doit cesser d'émettre sur une connexion chaque fois que la connexion expire, même si la fermeture en attente a été envoyée. Le multiplicateur doit être tel que représenté dans le Tableau 35.

Tableau 35 – Multiplicateur de temporisation

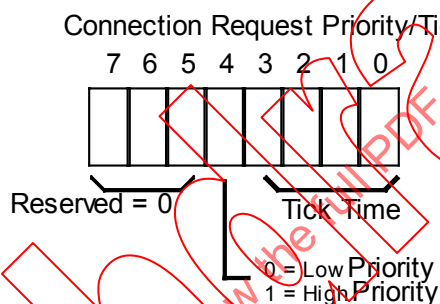
Valeur	Multiplicateur	Notes
0	X 4	Valeur par défaut
1	X 8	
2	X 16	
3	X 32	
4	X 64	
5	X 128	
6	X 256	
7	X 512	
8 à 255	reserved (réservé)	

4.1.6.4 Connection request priority and tick time (Priorité de demande et temps de top d'horloge d'une connexion)

Deux valeurs sont condensées en ce champ. La priorité de demande de connexion (connection request priority) détermine la priorité des messages de l'UCMM utilisés pour établir la connexion et n'a pas de corrélation avec la priorité de la connexion réalisée. Bien que la faible priorité (Low priority) soit utilisée pour la plupart des établissements de connexions, la haute priorité (High priority) peut être utilisée pour les circonstances spéciales qui exigent qu'une connexion soit réalisée rapidement. L'utilisation de la faible priorité a été choisie pour éviter l'engorgement avec la messagerie à temps critique. Les messages de l'UCMM ne peuvent pas utiliser la priorité programmée.

Le temps de top d'horloge (tick time) doit être utilisé conjointement à la valeur des tops d'horloge de temporisation de demande de connexion. Cette valeur détermine le temps entre les "ticks" (tops d'horloge) dans le dernier champ. Ainsi, si le temps d'horloge est 1 ms, une valeur de temporisation de demande de connexion de 5 se traduirait en 5 ms. Si le temps d'horloge était 2 ms, une valeur de temporisation de demande de connexion de 5 se traduirait en 10 ms.

Le format du Connection Request Priority/Tick Time doit être tel que montré à la Figure 5.



Légende

Anglais	Français
Connection Request Priority/Tick Time	Priorité de demande de connexion/Temps de top
Reserved = 0	Réservés
Tick Time	Temps de tops d'horloge
0 = Low Priority	Faible Priorité
1 = High Priority	Haute Priorité

Figure 5 – Temps de top d'horloge

Le Tableau 36 doit déterminer le temps entre tops.

Tableau 36 – Unités de top d'horloge

Temps de top		
Temps de top (binaire)	Temps par top	Temps max.
0000	1 ms	255 ms
0001	2 ms	510 ms
0010	4 ms	1 020 ms

Temps de top		
Temps de top (binaire)	Temps par top	Temps max.
0011	8 ms	2 040 ms
0100	16 ms	4 080 ms
0101	32 ms	8 160 ms
0110	64 ms	16 320 ms
0111	128 ms	32 640 ms
1000	256 ms	65 280 ms
1001	512 ms	130 560 ms
1010	1 024 ms	261 120 ms
1011	2 048 ms	522 240 ms
1100	4 096 ms	1 044 480 ms
1101	8 192 ms	2 088 960 ms
1110	16 384 ms	4 177 920 ms
1111	32 768 ms	8 355 840 ms

4.1.6.5 Tops de temporisation de demande de connexion

Les tops de temporisation de demande de connexion doivent être utilisés pour spécifier la quantité de temps que l'application émettrice doit attendre pour que la connexion soit établie. Chaque saut de la connexion a une valeur de temporisation spécifique à la liaison qui est mise en œuvre par l'UCMM et, après que les répétitions automatiques de tentatives automatiques ont été épuisées sans un acquittement, l'UCMM doit générer une erreur de temporisation. Par conséquent, si un dispositif dans le chemin doit être soit absent, soit trop occupé pour acquitter une demande de l'UCMM, une temporisation d'UCMM doit se produire aussi rapidement que possible. La seule fois que la période de temporisation de demande de connexion serait rencontrée doit être si le nœud-cible acquitte la demande de connexion, mais échoue à répondre ou si la demande de connexion doit être délivrée, mais une défaillance dans le réseau empêche que la réponse soit retournée. La temporisation de demande de connexion ne doit pas être la valeur de temporisation de chemin dérivée du CM_RPI et utilisée pour fermer une connexion après qu'elle doit être établie. La temporisation de demande de connexion doit être spécifiée par une combinaison des tops de temporisation de demande de connexion multipliés par le temps par top d'horloge qui doit être déterminé par la valeur du temps de top.

Sachant qu'une temporisation de demande de connexion particulière peut être exprimée en plusieurs formats différents, la règle doit être de **toujours le condenser de sorte que la valeur du temps d'horloge doit être réduite au maximum**. Cela autorise la meilleure granularité pour diminuer la temporisation à mesure que le message est transmis par sauts. Cela signifie aussi que la valeur du top d'horloge peut varier à mesure que le message est transmis par sauts multiples. À titre d'exemple, le codage de 300 ms peut être:

1. 0001 10010110 -> Temps par top = 2 ms, Tops = 150 (correct)
2. 0010 01001011 -> Temps par top = 4 ms, Tops = 75 (erroné)

La valeur de temporisation de demande de connexion doit être fonction du nombre de segments de port dans le chemin de connexion, du temps de traitement au nœud-cible et de l'engorgement des réseaux intermédiaires. La temporisation de demande de connexion peut être une valeur relativement élevée, car la pleine temporisation de demande de connexion ne

doit se produire qu'en quelques rares cas. La seule fois que se produirait la pleine temporisation de demande de connexion serait si:

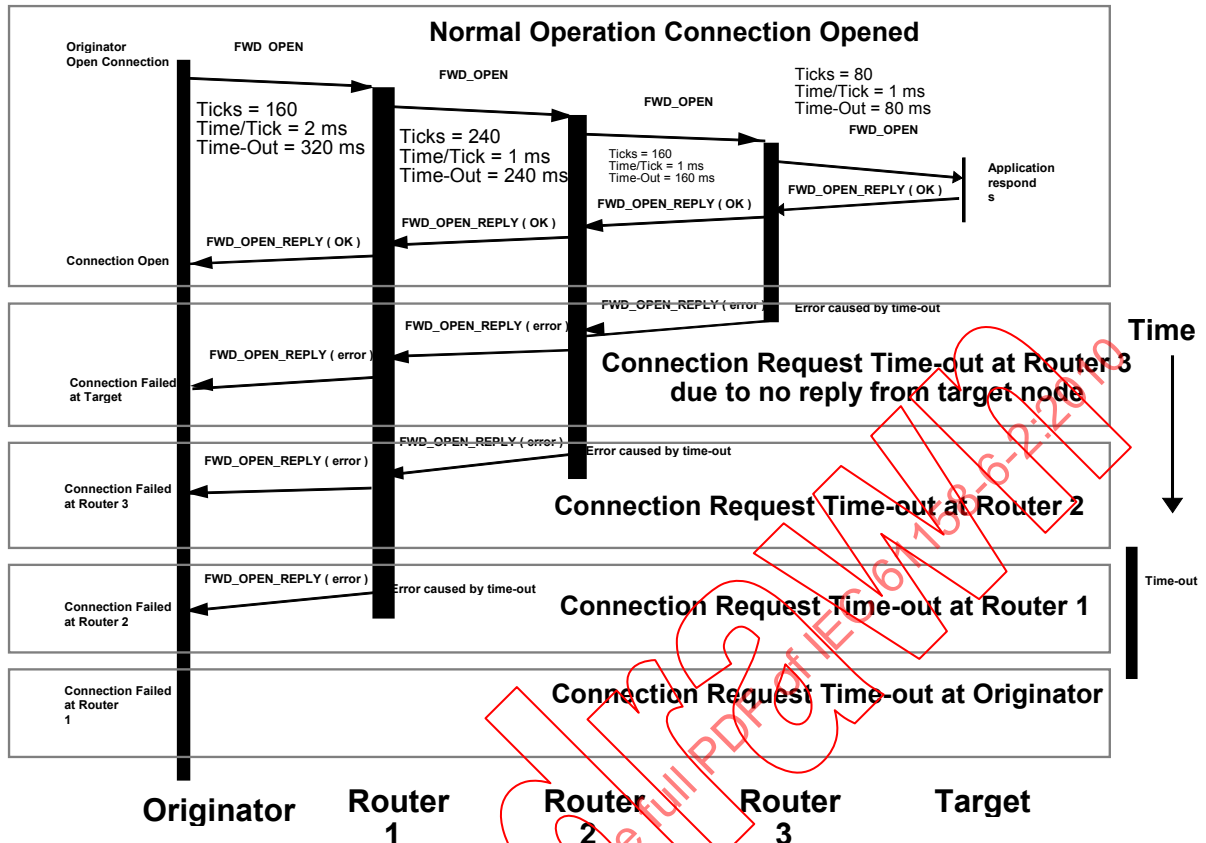
- la demande de connexion avait été délivrée avec succès au nœud-cible et le nœud avait échoué à générer une réponse. Cela serait habituellement provoqué par une erreur de firmware dans le dispositif.
- la demande de connexion avait été délivrée avec succès au nœud-cible et la réponse avait été rejetée en raison d'un engorgement au nœud intermédiaire.
- la demande de connexion avait été délivrée avec succès au nœud-cible et le nœud-cible ou l'un des nœuds intermédiaires avait été retiré ou mis hors tension avant que la réponse ne fût retournée.

L'UCMM doit être utilisé pour transmettre la demande de connexion vers le prochain nœud du chemin. L'UCMM envoie la demande et attend un acquittement ou une réponse pendant un temps spécifique à la liaison; si aucun acquittement ou aucune réponse ne doit être reçu(e) avant la temporisation spécifique à la liaison, la demande doit être répétée. Après un nombre (trois en général) de répétitions de tentatives spécifique à la liaison, l'UCMM doit signaler une temporisation sur la demande de connexion. Le Connection Manager qui a généré la demande doit retourner une réponse d'erreur à la demande de connexion. Par conséquent, la demande de connexion doit s'arrêter aussitôt que possible pour les défauts de système les plus courants

- dispositifs absents ou mis hors tension dans le chemin;
- engorgement au dispositif dans le chemin.

Pour aider au débogage des systèmes, chaque saut dans le chemin doit décrémenter la temporisation de 80 ms avant de l'expédier dans le Forward_Open vers le prochain dispositif du chemin. Chaque dispositif intermédiaire (routeur) doit arrondir la valeur de temporisation de connexion qui doit être décrémentée de sorte qu'un minimum de 1 top doit être soustrait. Cela doit permettre au dernier dispositif avant la défaillance de temporiser le premier et retourner les informations relatives au point en lequel la connexion a été faite avant la défaillance. Ces informations peuvent être utilisées pour aider au débogage de problèmes dans un système. Une fonction système typique est montrée à la Figure 6.

Connection Request Time-out in Multi-hop Systems



Légende

Anglais	Français
Application responds	L'application répond.
Connection Failed at Router 1	Échec connexion au niveau du routeur 1
Connection Failed at Router 2	Échec connexion au niveau du routeur 2
Connection Failed at Router 3	Échec connexion au niveau du routeur 3
Connection Failed at Target	Échec connexion au niveau de la cible
Connection Open	Connexion ouverte
Connection Request Time-out at Originator	Temporisation de demande de connexion au niveau de l'émetteur
Connection Request Time-out at Router 1	Temporisation de demande de connexion au niveau du routeur 1
Connection Request Time-out at Router 2	Temporisation de demande de connexion au niveau du routeur 2
Connection Request Time-out at Router 3 due to no reply from target node	Temporisation de demande de connexion au niveau du routeur 3 en raison de l'absence de réponse de la part du noeud-cible
Connection Request Time-out in Multi-hop Systems	Temporisation de demande de connexion dans les systèmes à sauts multiples
Error caused by time-out	Erreur pour cause de temporisation
Normal Operation Connection Opened	Fonctionnement normal connexion ouverte
Open Connection	Ouvrir Connexion
Originator	Émetteur

Anglais	Français
Originator Router	Routeur émetteur
Router	Routeur
Target	Cible
Ticks = 160	Tops = 160
Ticks = 240	Tops = 240
Ticks = 80	Tops = 80
Time	Temps
Time/Tick = 1 ms	Time/Tick = 1 ms
Time/Tick = 2 ms	Temps/Top = 2 ms
Time-out	Temporisation
Time-Out = 160 ms	Temporisation = 160 ms
Time-Out = 240 ms	Temporisation = 240 ms
Time-Out = 320 ms	Temporisation = 320 ms
Time-Out = 80 ms	TimTemporisation = 80 ms

Figure 6 – Temporisation d'établissement de connexion

À la Figure 6, une connexion est demandée sur un chemin contenant 4 sauts. La Connection request Time-out est spécifiée comme étant de 320 ms par l'émetteur de la connexion. Dans le premier cas, la connexion est parachevée et les valeurs de temporisation calculées sont montrées en chaque saut dans le chemin. Le deuxième cas est celui où le dernier dispositif reçoit un acquittement, mais ne reçoit pas de réponse et temporise. Il retourne une erreur de temporisation à l'émetteur. Comme la valeur de temporisation décroît pour chaque saut, l'erreur de temporisation doit être retournée à l'émetteur avant que l'émetteur ne temporise.

Les autres cas montrent des temporisations en les autres dispositifs et dans chaque cas, l'erreur de temporisation doit être retournée à l'émetteur avant que l'émetteur lui-même ne temporise. Le dernier cas est celui où aucune réponse n'est reçue par l'émetteur et le processus d'ouverture de connexion a expiré. La raison de faire décroître la temporisation en chaque nœud est que peuvent être ainsi envoyées des informations d'erreur supplémentaires indiquant à quel endroit dans le chemin la temporisation a eu lieu. Se rappeler que si l'un quelconque des dispositifs ne reçoit pas un acquittement, la demande de connexion doit immédiatement être temporisée et une réponse d'erreur doit être générée.

4.1.6.6 Connection serial number

Le numéro de série de connexion (connection serial number) doit être une valeur unique de 16 bits sélectionnée par le Connection Manager au niveau de l'émetteur de la connexion. L'émetteur doit s'assurer que la valeur de 16 bits est unique pour le dispositif. Aucune autre signification ne doit être placée sur le numéro par un quelconque autre nœud dans le chemin de connexion. Les numéros de série de connexion doivent être uniques, mais peuvent ne pas être séquentiels. Par exemple, une interface opérateur peut avoir un grand nombre de connexions ouvertes en même temps, chacune avec un numéro unique. Les mêmes valeurs pourraient être répétées en d'autres postes de l'interface opérateur. Une mise en œuvre possible serait d'avoir une liste de connexions qui pointe vers le descripteur pour chaque connexion et le numéro de série de connexion pourrait être l'indice dans le tableau.

4.1.6.7 Vendor ID

Le numéro de vendeur doit être un nombre unique attribué aux divers vendeurs de produits. Chaque vendeur a un numéro unique attribué.

NOTE Les ID de vendeur sont attribués par ODVA, Inc.

4.1.6.8 Originator serial number

Le numéro de série d'émetteur (originator serial number) doit être une valeur unique de 32 bits qui est attribuée à un dispositif au moment de la fabrication. Cette valeur doit être garantie comme étant unique pour tous les dispositifs fabriqués par le même vendeur. Aucune signification ne doit être attachée au numéro. La combinaison de l'ID de vendeur et de l'Originator Serial Number doit être unique dans le système tout entier.

4.1.6.9 Connection number

Le numéro de connexion (connection number) doit être une valeur de 16 bits qui est attribuée par le Connection Manager lorsqu'une connexion est ouverte. Cette valeur permet à d'autres nœuds d'obtenir des données de connexion délivrées par le Connection Manager. Le numéro ne doit pas être confondu avec le Connection Serial Number (numéro de série de connexion).

4.1.6.10 Connection path size

La longueur de chemin de connexion (connection path size) doit être la longueur du chemin de connexion en mots de 16 bits. La longueur du chemin de connexion varie au cours du processus de connexion, car chaque nœud sur le chemin de connexion enlève le segment de port courant et transmet seulement les segments de chemin restants vers le prochain nœud.

4.1.6.11 Connection path (chemin de connexion)

Se référer à la description de Path (chemin) en 4.1.8.9.

4.1.6.12 Connection remaining path (chemin de connexion restant)

Se référer à la description de Path (chemin) en 4.1.8.9.

4.1.6.13 Network connection ID

L'ID de connexion réseau (Network Connection ID) doit être une valeur de 32 bits construite comme suit:

- Les bits 31-24 doivent être réservés et doivent être mis à zéro;
- Les bits 23-16 doivent contenir un node MAC ID dépendant du type de connexion sélectionné;
- les bits 15-0 doivent contenir le numéro de connexion de 16 bits.

Le Tableau 37 montre des options pour la sélection de l'ID de connexion.

Tableau 37 – Sélection de l'ID de connexion

Connection Type (type de connexion)	MAC ID	Connection number
Multipoint	MAC du producteur	Unique par dispositif
Point à point	MAC du consommateur	Unique par dispositif

L'ID de connexion réseau (Network Connection ID) doit être spécifique à une liaison et ne doit pas être relié au numéro de série de connexion, qui est spécifique à une connexion et le même sur toutes les liaisons. Les champs du Network Connection ID doivent être utilisés pour établir le mécanisme de filtrage pour la liaison spécifiée.

L'étiquette générique pour les Lpackets échangés sur une connexion et décrits dans la CEI 61158-4-2:2010 doit être construite comme suit, en fonction du contenu des champs CID:

- le premier octet émis de l'étiquette générique doit contenir le MAC ID;
- Le deuxième octet émis de l'étiquette générique doit contenir l'octet de poids faible du numéro de connexion;
- Le troisième octet émis de l'étiquette générique doit contenir l'octet de poids fort du numéro de connexion;

Pour garantir que les ID de connexion sont uniques sur une liaison, les règles suivantes doivent être utilisées pour choisir les CID:

- le producteur doit choisir le CID pour un transport multipoint;
- le consommateur doit choisir le CID pour un transport point à point;
- sur une liaison CP 2/1, le nœud chargé du choix du CID (soit le producteur, soit le consommateur) doit insérer son MAC ID dans les bits 23-16 du CID du Forward_Open;
- un CID multipoint ne doit pas être réutilisé tant que les connexions associées au CID n'ont pas été fermées ou temporisées;
- le fait de recevoir un paquet à étiquette fixe "I'm alive" issu d'un nœud doit fermer immédiatement toutes les connexions avec le dispositif en question.

4.1.6.14 Transport class and trigger

La classe de transport et le déclencheur (transport class and trigger) spécifient le type de transport requis pour la connexion. Ces informations ne doivent pas être utilisées par le Connection Manager, mais doivent être transmises vers l'application. L'application doit déterminer si le type de transport est pris en charge et si une instance du transport requis est disponible. L'octet qui code la classe de transport et le déclencheur doit avoir la forme suivante montrée dans le Tableau 38.

Tableau 38 – Format de classe de transport, déclencheur et Is_Server

TransportClass (4 bits)	TriggerMode (3 bits)	Is_Server (1 bit)
NULL = 0 DUPLICATE_DETECT = 1 ACKNOWLEDGED = 2 VERIFIED = 3 NONBLOCKING = 4 FRAGMENTING = 5 MULTIPOINT_FRAG = 6	CYCLIC = 0 CHANGE_OF_STATE = 1 APPLICATION = 2	IS_NOT_SERVER = 0 IS_SERVER = 1

Les 4 bits de poids faible, `class` (classe), doivent déterminer quel type de transport est spécifié et doivent être situés dans la plage de 0 à 6. Les valeurs de `class` 7 à 15 doivent être réservées. Les bits 4 à 6, appelés `trigger` (déclencheur), doivent déterminer quel événement provoque la production de données sur la connexion et doivent être situés dans la plage de 0 à 2. Les valeurs de `trigger` 3 à 7 doivent être réservées. Pour les transports qui ont besoin de spécifier la cible comme un serveur, le champ `is_server` doit être 1; autrement, il doit être 0. Pour les classes de transport 0 et 1, le champ `is_server` doit être ignoré. Le bit 7 doit être le champ `is_server`;

Voir la CEI 61158-5-2:2010 pour des détails sur le comportement pour les différentes combinaisons de `is_server`, `class` et `trigger`, `CM_RPI` et `CM_API`.

4.1.7 En-têtes de MR

Tous les paquets de demande livrés au Message Router, soit en provenance d'une connexion, soit en provenance de l'UCMM, doivent avoir un en-tête de la forme montrée dans le Tableau 39.

Tableau 39 – Format de MR_Request_Header

Paramètre	Format
Service	USINT
path_size	USINT
path[]	UINT

Le premier octet, service, doit être livré directement à l'objet d'application et doit se situer dans la plage de 1 à 127. Le path_size doit déterminer le nombre de mots de 16 bits contenus dans la matrice path. Le path doit contenir les informations d'adressage pour le paquet et doit être bourré.

Les paquets de réponse issus du Message Router doivent avoir un en-tête de la forme montrée dans le Tableau 40.

Tableau 40 – Format de MR_Response_Header

Paramètre	Format
Service	USINT
Reserved	USINT
general_status	USINT
Extended_status_size	USINT
Extended_status[]	WORD

Le champ service doit être la valeur du champ MR_Request_Header.service avec son bit de poids fort mis. Si MR_Request_Header.service est égal à 0x05, le champ MR_Response_Header.service doit être égal à 0x85. Le champ general_status doit être rempli avec l'aide du paramètre Status Code de la réponse de service. De même, les paramètres extended_status, extended_status_size, et response doivent respectivement être remplis avec l'aide du paramètre Extended Status et autres paramètres de réponse issus de la réponse de service. Le response_size ne doit pas être inclus de façon explicite dans le MR_Response_Header; Il doit être sous-entendu par le nombre d'octets dans MR_Response_Header.

4.1.8 OM_Service_PDU

4.1.8.1 Syntaxe générale

4.1.8.1.1 Get_Attribute_All

Le corps de Get_Attribute_All_RequestPDU est vide.

Le corps de Get_Attribute_All_ResponsePDU doit être tel que spécifié dans le Tableau 41.

Tableau 41 – Structure du corps de Get_Attribute_All_ResponsePDU

Nom	Type de donnée
Attribute Data (Données d'attribut)	Spécifique à un attribut d'objet/de classe

4.1.8.1.2 Set_Attribute_All

Le corps de Set_Attribute_All_RequestPDU doit être tel que spécifié dans le Tableau 42.

Tableau 42 – Structure du corps de Set_Attribute_All_RequestPDU

Nom	Type de donnée
Attribute Data (Données d'attribut)	Spécifique à un attribut d'objet/de classe

Le corps de Get_Attribute_All_ResponsePDU est vide.

4.1.8.1.3 Get_Attribute_List

Le corps de Get_Attribute_List_RequestPDU doit être tel que spécifié dans le Tableau 43.

Tableau 43 – Structure du corps de Get_Attribute_List_RequestPDU

Nom	Type de donnée
Attribute Count (compte d'attributs)	UINT
Attribute List (liste d'attributs)	Array of UINT's

Le corps de Get_Attribute_List_ResponsePDU doit être tel que spécifié dans le Tableau 44.

Tableau 44 – Structure du corps de Get_Attribute_List_ResponsePDU

Nom	Type de donnée	Sémantique des valeurs (Semantics of values)
Attribute Count (compte d'attributs)	UINT	
Attribute Data (Données d'attribut)	Array of STRUCT	
Attribute Identifier (identificateur d'attribut)	UINT	
Attribute Status (statut d'attribut)	UINT	Les valeurs 0 à 255 sont réservées à des codes de statut de services communs mineurs. Les valeurs 256 à 65 535 sont disponibles pour les erreurs spécifiques à un attribut d'objet/de classe.
Attribute Value (valeur d'attribut)	Spécifique à un attribut d'objet/de classe	

4.1.8.1.4 Set_Attribute_Lists

Le corps de Set_Attribute_List_RequestPDU doit être tel que spécifié dans le Tableau 45.

Tableau 45 – Structure du corps de Set_Attribute_List_RequestPDU

Nom	Type de donnée
Attribute Count (compte d'attributs)	UINT
Attribute Data (Données d'attribut)	Array of STRUCT
Attribute Identifier (identificateur d'attribut)	UINT
Attribute Value (valeur d'attribut)	Spécifique à un attribut d'objet/de classe

Le corps de Set_Attribute_List_ResponsePDU doit être tel que spécifié dans le Tableau 46.

Tableau 46 – Structure du corps de Set_Attribute_List_ResponsePDU

Nom	Type de donnée	Sémantique des valeurs (Semantics of values)
Attribute Count (compte d'attributs)	UINT	
Attribute Status List (liste de statuts d'attribut)	Array of STRUCT	
Attribute Identifier (identificateur d'attribut)	UINT	
Attribute Status (statut d'attribut)	UINT	Les valeurs 0 à 255 sont réservées à des codes de statut de services communs mineurs. Les valeurs 256 à 65 535 sont disponibles pour les erreurs spécifiques à un attribut d'objet/de classe.

4.1.8.1.5 Reset (réinitialisation)

Le corps de Reset_RequestPDU doit être tel que spécifié dans le Tableau 47, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Reset est spécifié. Sinon, il est vide.

Tableau 47 – Structure du corps de Reset_RequestPDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Reset_ResponsePDU doit être tel que spécifié dans le Tableau 48, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Reset est spécifié. Sinon, il est vide.

Tableau 48 – Structure du corps de Reset_ResponsePDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.6 Start

Le corps de Start_RequestPDU doit être tel que spécifié dans le Tableau 49, si le paramètre de service **facultatif** “Object Specific Data” (données spécifiques à un objet) de la demande Start est spécifié. Sinon, il est vide.

Tableau 49 – Structure du corps de Start_RequestPDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Start_ResponsePDU doit être tel que spécifié dans le Tableau 50, si le paramètre de service **facultatif** “Object Specific Data” (données spécifiques à un objet) de la réponse Start est spécifié. Sinon, il est vide.

Tableau 50 – Structure du corps de Start_ResponsePDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.7 Stop

Le corps de Stop_RequestPDU doit être tel que spécifié dans le Tableau 51, si le paramètre de service **facultatif** “Object Specific Data” (données spécifiques à un objet) de la demande Stop est spécifié. Sinon, il est vide.

Tableau 51 – Structure du corps de Stop_RequestPDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Stop_ResponsePDU doit être tel que spécifié dans le Tableau 52, si le paramètre de service **facultatif** “Object Specific Data” (données spécifiques à un objet) de la réponse Stop est spécifié. Sinon, il est vide.

Tableau 52 – Structure du corps de Stop_ResponsePDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.8 Create

Le corps de Create_RequestPDU doit être tel que spécifié dans le Tableau 53, si le paramètre de service **facultatif** “Object Specific Data” (données spécifiques à un objet) de la demande Create est spécifié. Sinon, il est vide.

Tableau 53 – Structure du corps de Create_RequestPDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Create_ResponsePDU doit être tel que spécifié dans le Tableau 54, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Create est spécifié. Sinon, il est vide.

Tableau 54 – Structure du corps de Create_ResponsePDU

Nom	Type de donnée
Instance ID	UINT
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.9 Delete

Le corps de Delete_RequestPDU doit être tel que spécifié dans le Tableau 55, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Delete est spécifié. Sinon, il est vide.

Tableau 55 – Structure du corps de Delete_RequestPDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Delete_ResponsePDU doit être tel que spécifié dans le Tableau 56, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Delete est spécifié. Sinon, il est vide.

Tableau 56 – Structure du corps de Delete_ResponsePDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.10 Get_Attribute_Single

Le corps de Get_Attribute_Single_RequestPDU est vide.

Le corps de Get_Attribute_Single_ResponsePDU doit être tel que spécifié dans le Tableau 57.

Tableau 57 – Structure du corps de Get_Attribute_Single_ResponsePDU

Nom	Type de donnée
Attribute Data (Données d'attribut)	Spécifique à un attribut d'objet/de classe

4.1.8.1.11 Set_Attribute single

Le corps de Set_Attribute_Single_RequestPDU doit être tel que spécifié dans le Tableau 58.

Tableau 58 – Structure du corps de Set_Attribute_Single_RequestPDU

Nom	Type de donnée
Attribute Data (Données d'attribut)	Spécifique à un attribut d'objet/de classe

Le corps de Set_Attribute_Single_ResponsePDU doit être tel que spécifié dans le Tableau 59, si le paramètre de service **facultatif** “Object Specific Data” (données spécifiques à un objet) de la réponse Set_Attribute_Single est spécifié. Sinon, il est vide.

Tableau 59 – Structure du corps de Set_Attribute_Single_ResponsePDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.12 Find_Next_Object_Instance

Le corps de Find_Next_Object_Instance_RequestPDU doit être tel que spécifié dans le Tableau 60.

Tableau 60 – Structure du corps de Find_Next_Object_Instance_RequestPDU

Nom	Type de donnée
Maximum Returned values (valeurs maximales retournées)	USINT

Le corps de Find_Next_Object_Instance_ResponsePDU doit être tel que spécifié dans le Tableau 61.

Tableau 61 – Structure du corps de Find_Next_Object_Instance_ResponsePDU

Nom	Type de donnée
Number Of List Members (nombre des éléments de liste)	USINT
Instance ID List (liste des ID d'instance)	Array of UINT

4.1.8.1.13 NOP

Le corps de NOP_RequestPDU est vide.

Le corps de NOP_ResponsePDU est vide.

4.1.8.1.14 Apply_Attributes

Le corps de Apply_Attributes_RequestPDU doit être tel que spécifié dans le Tableau 62, si le paramètre de service **facultatif** “Object Specific Data” (données spécifiques à un objet) de la demande Apply_Attributes est spécifié. Sinon, il est vide.

Tableau 62 – Structure du corps de Apply_Attributes_RequestPDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Apply_Attributes_ResponsePDU doit être tel que spécifié dans le Tableau 63, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Apply_Attributes est spécifié. Sinon, il est vide.

Tableau 63 – Structure du corps de Apply_Attributes_ResponsePDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.15 Save

Le corps de Save_RequestPDU doit être tel que spécifié dans le Tableau 64, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Save est spécifié. Sinon, il est vide.

Tableau 64 – Structure du corps de Save_RequestPDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Save_ResponsePDU doit être tel que spécifié dans le Tableau 65, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Save est spécifié. Sinon, il est vide.

Tableau 65 – Structure du corps de Save_ResponsePDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.16 Restore

Le corps de Restore_RequestPDU doit être tel que spécifié dans le Tableau 66, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la demande Restore est spécifié. Sinon, il est vide.

Tableau 66 – Structure du corps de Restore_RequestPDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Restore_ResponsePDU doit être tel que spécifié dans le Tableau 67, si le paramètre de service **facultatif** "Object Specific Data" (données spécifiques à un objet) de la réponse Restore est spécifié. Sinon, il est vide.

Tableau 67 – Structure du corps de Restore_ResponsePDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

4.1.8.1.17 Group_Sync

Le corps de Group_Sync_RequestPDU doit être tel que spécifié dans le Tableau 68.

Tableau 68 – Structure du corps de Group_Sync_RequestPDU

Nom	Type de donnée
Object Specific Data	Spécifique à un service d'objet/de classe

Le corps de Group_Sync_ResponsePDU doit être tel que spécifié dans le Tableau 69.

Tableau 69 – Structure du corps de Group_Sync_ResponsePDU

Nom	Type de donnée	Sémantique des valeurs (Semantics of values)
IsSynchronized	BOOL	Indique si l'objet est synchronisé à la Base de temps PTP 0 = Not synchronized (c'est-à-dire: non synchronisé) 1 = Is synchronized (c'est-à-dire: est synchronisé).
Object Specific Data	Spécifique à un service d'objet/de classe	Spécifique à un service d'objet/de classe

4.1.8.2 Éléments de syntaxe spécifiques à un object Identity**4.1.8.2.1 Attributs****4.1.8.2.1.1 Attributs de classe**

Le format des attributs de classe d'objets Identity doit être tel que spécifié dans le Tableau 70.

Tableau 70 – Attributs de classe d'objets Identity

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
2	Max Instance	UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.2.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Identity est tel que spécifié dans le Tableau 71.

Tableau 71 – Attributs d'instance d'objet Identity

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Vendor ID	UINT	La valeur zéro ne doit pas être utilisée.

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
2	Device Type	UINT	Une énumération des plages de Device Type peut être consultée en 4.1.10.2.5.
3	Product Code	UINT	La classe zéro n'est pas valide.
4	Revision	STRUCT of	La valeur zéro ne doit être valide pour aucun des champs Major Revision et Minor Revision d'un produit conforme. Un produit endommagé peut retourner zéro pour indiquer une révision inconnue si son stockage de révision devient corrompu.
	Major Revision	USINT	L'attribut Major Revision (révision majeure) doit être limité à 7 bits. Le huitième est réservé et doit être zéro.
	Minor Revision	USINT	
5	Status	WORD	Les définitions des bits sont décrites dans le Tableau 72.
6	Serial Number (Numéro de série)	UDINT	
7	Product Name	SHORT_STRING	Le nombre maximal de caractères de 8 bits dans cette chaîne doit être 32. Chacun des caractères doit se situer dans la plage de 0x20 à 0x7E.
8	État	USINT	Présent état du dispositif tel que représenté par le diagramme de transition d'états: 0 = Nonekistent 1 = Device Self Testing 2 = Standby 3 = Operational 4 = Major Recoverable Fault 5 = Major Unrecoverable Fault 6 à 254 = Reserved 255 = Valeur par défaut pour le service Get_Attributes_All
9	Configuration Consistency Value	UINT	Le contenu identifie la configuration du dispositif.
10	Heartbeat Interval	USINT	L'intervalle nominal entre les messages "heartbeat", en secondes.
11	Active Language	STRUCT of	Langue actuellement active pour le dispositif
	Language1	USINT	Le champ language1 issu du type de données STRINGI.
	Language2	USINT	Le champ language2 issu du type de données STRINGI.
	Language3	USINT	Le champ language3 issu du type de données STRINGI.
12	Supported Language	ARRAY of STRUCT of	Liste des langues prises en charge par les chaînes de caractères du type de données STRINGI au sein du dispositif
	Language1	USINT	Le champ language1 issu du type de données STRINGI.
	Language2	USINT	Le champ language2 issu du type de données STRINGI.
	Language3	USINT	Le champ language3 issu du type de données STRINGI.
13	International Product Name	STRINGI	Noms du produit dans diverses langues
14	Semaphore	STRUCT of	Access Synchronization Semaphore (sémaphore de synchronisation d'accès)

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
	Vendor Number	UINT	Par défaut: Toutes les valeurs nulles
	Client Serial Number	UDINT	
	Semaphore Timer	ITIME	Plage valide du temporisateur 100 à 32 767
15	Assigned_Name	STRING1	Nom attribué par l'utilisateur
16	Assigned_Description	STRING1	Description attribuée par l'utilisateur
17	Geographic_Location	STRING1	Emplacement attribué par l'utilisateur

Tableau 72 – Définitions de bits de l'objet Identity pour l'attribut d'instance de statut

Bit(s)	Appelé	Définition
0	Owned (possédé)	1 pour TRUE, 0 pour FALSE
1		Réservé, mis à 0
2	Configured (configuré)	1 pour TRUE, 0 pour FALSE
3		Réservé, mis à 0
4,5,6,7		Voir Tableau 73
8	Minor Recoverable Fault (Défaut récupérable mineur)	1 pour TRUE, 0 pour FALSE
9	Minor Unrecoverable Fault (Défaut irrécupérable mineur)	1 pour TRUE, 0 pour FALSE
10	Major Recoverable Fault (Défaut récupérable majeur)	1 pour TRUE, 0 pour FALSE
11	Major Unrecoverable Fault (Défaut irrécupérable majeur)	1 pour TRUE, 0 pour FALSE
12, 13		Réservés, doivent être mis à 0.
14, 15		Réservés, doivent être mis à 0.

Les valeurs des bits de statut 4 à 7 doivent être telles que montrées dans le Tableau 73.

Tableau 73 – Bits 4 – 7 de l'attribut d'instance de statut

Bits 4, 5, 6, 7:	Signification
0 0 0 0	En autotest (mise sous tension) ou état inconnu
0 0 0 1	Mise à jour de firmware en cours
0 0 1 0	Défaut de communication (connexion perdue)
0 0 1 1	Déverrouillé, en attente de connexion
0 1 0 0	Mauvaise configuration non volatile
0 1 0 1	Défaut majeur (voir bits 10 et 11 pour la classification des défauts et les états des DEL) Un défaut mineur n'est pas un changement d'état.
0 1 1 0	Connecté, Actif
0 1 1 1	Au repos (type de mode de programme)
1 0 0 0 1 0 0 1	réservé, ne doit pas être utilisé

ID d'attribut	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	13	Serial Number (octet de poids fort)							
7	14	Product Name length (longueur du nom de produit)							
	15	Product Name (1 ^{er} caractère)							
	16	Product Name (2 ^e caractère)							
	n	Product Name (dernier caractère)							

La longueur du nom n'étant pas connue avant l'émission de la demande de service Get_Attribute_All, les réalisateurs doivent allouer assez d'espace mémoire pour stocker une réponse ayant une longueur pouvant atteindre 32 caractères.

Pour maintenir la cohérence, si des attributs d'instance sont ajoutés à une date ultérieure, la réponse Get_Attribute_All au niveau instance doit être modifiée pour inclure les attributs définis dans le Tableau 76.

Tableau 76 – Données de réponse modifiée spécifiques à un service/objet de niveau instance pour Get_Attribute_All

ID d'attribut	Octet	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
8	n+1	State Default = 255							
9	n+2	Espace réservé à l'octet de poids faible de l'attribut 9, valeur par défaut = 0							
	n+3	Espace réservé à l'octet de poids fort de l'attribut 9, valeur par défaut = 0							
10	n+4	Intervalle Heartbeat, Valeur par défaut = 0							

4.1.8.2.2.2 Service Reset

Le service commun Reset doit avoir le paramètre spécifique à un objet spécifié dans le Tableau 77.

Tableau 77 – Paramètre spécifique à un objet pour Reset

Nom	Type	Sémantique des valeurs
Type	USINT	La valeur 0 doit émuler aussi étroitement que possible la puissance en cycle sur l'élément que l'objet Identity représente. Cette valeur doit être la valeur par défaut si ce paramètre est omis. La valeur 1 doit retourner aussi étroitement que possible à la configuration "out-of-box" (au déballage), puis doit émuler la puissance en cycle aussi étroitement que possible. La valeur 2 doit retourner aussi étroitement que possible à la configuration "out-of-box" avec l'exception des paramètres de liaison de communication, puis émuler la puissance en cycle aussi étroitement que possible. Les paramètres de liaison de communication qui sont à préserver sont définis par chaque type de réseau. Les valeurs 3 à 255 sont réservées.

4.1.8.3 Eléments de syntaxe spécifique à l'objet Message Router

4.1.8.3.1 Attributs

4.1.8.3.1.1 Attributs de classe

Les attributs de classe d'objets Message Router doivent être tels que spécifiés dans le Tableau 78.

Tableau 78 – Attributs de classe d'objets Message Router

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
4	Optional attribute list	STRUCT of	
	Nombre d'attributs	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.3.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Message Router est tel que spécifié dans le Tableau 79.

Tableau 79 – Attributs d'instance d'objet Message Router

ID d'attribut	Nom	Type de donnée
1	Object_List	STRUCT of
	Number	UINT
	Classes	ARRAY of UINT
2	Number available	UINT

4.1.8.3.2 Services communs

Réponse Get_Attribute_All

Au **niveau de la classe**, l'ordre des attributs retournés dans le paramètre "Attribute Data" de la réponse Get_Attribute_All doit tel que spécifié dans le Tableau 80. Les valeurs par défaut pour tous les attributs de classe non pris en charge doivent être telles que montrées dans le tableau en question.

Tableau 80 – Données de réponse spécifiques à un service/objet de niveau classe pour Get_Attribute_All

ID d'attribut de classe	Nom d'attribut, description et valeur par défaut
1	Revision valeur par défaut = 1
4	Optional attribute list, nombre d'attributs, valeur par défaut = 0

ID d'attribut de classe	Nom d'attribut, description et valeur par défaut
5	Optional service list, nombre de services, valeur par défaut = 0
6	Max ID, nombre d'attributs de classe, valeur par défaut = 0
7	Max ID, nombre d'attributs d'instance, valeur par défaut = 0

Les valeurs par défaut pour tous les attributs d'instance non pris en charge doivent être telles que montrées dans le Tableau 81

Tableau 81 – Données de réponse spécifiques à un service/objet de niveau instance pour Get_Attribute_All

ID d'attribut d'instance	Nom d'attribut, description et valeur par défaut
1	Object list number, nombre de classes prises en charge, valeur par défaut = 0
2	Number available, nombre maximum de connexions, valeur par défaut = 0
3	Toujours = 0x0000

4.1.8.4 Éléments de syntaxe spécifiques à un objet Assembly

4.1.8.4.1 Attributs

4.1.8.4.1.1 Attributs de classe

Le format des attributs de classe d'objets Assembly doit être tel que spécifié dans le Tableau 82.

Tableau 82 – Attributs de classe d'objets Assembly

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 2
2	Max Instance	UINT	

4.1.8.4.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Assembly est tel que spécifié dans le Tableau 83.

Tableau 83 – Attributs d'instance d'objet Assembly

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Number of Members in List	UINT	
2	Member List	ARRAY of STRUCT	Cet attribut à un type de données complexe.
	Member Data Size	UINT	Taille en bits
	Member Path Size	UINT	Taille en bits (0 = Empty Path «chemin vide»)
	Member Path	Packed EPATH	Chemin «condensé» vers les données de membre. Voir 4.1.8.9 pour la définition de Path

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
3	Data	ARRAY of SWORD	Contient toutes les données de membre condensées en une matrice. Ces données peuvent contenir plusieurs types de données différents. Pour l'efficacité, il vaut mieux maintenir ce mot de données aligné en le condensant sur les frontières des mots et en ajoutant un bourrage selon les besoins. Ceci peut être accompli en utilisant des "chemins vides" (Member Path Size = 0)
4	Size	UINT	

4.1.8.5 Éléments de syntaxe spécifiques à un objet Acknowledge Handler

4.1.8.5.1 Attributs

4.1.8.5.1.1 Attributs de classe

Le format des attributs de classe d'objets Acknowledge Handler doit être tel que spécifié dans le Tableau 84.

Tableau 84 – Attributs de classe d'objets Acknowledge Handler

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
4	Optional attribute list	STRUCT of	
	Number of attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	
	Optional services	ARRAY of UINT	
6	Max ID Number of Class Attributes	UINT	
7	Max ID Number of Instance Attributes	UINT	

4.1.8.5.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Acknowledge Handler est tel que spécifié dans le Tableau 85.

Tableau 85 – Attributs d'instance d'objet Acknowledge Handler

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Acknowledge Timer	UINT	Plage 1 ms à 65 535 ms (0 non valide) Valeur par défaut = 16
2	Retry Limit	USINT	Plage 0 à 255 Valeur par défaut = 1
3	COS Producing Connection Instance	UINT	Connection Instance ID (identificateur d'instance de connexion)
4	Ack List Size	SWORD	0 = Dynamique > 0 = Nombre max de membres
5	Ack List	STRUCT of	
		SWORD	Nombre de membres dans la matrice
		ARRAY of UINT	Liste d'identificateurs d'instance de connexion
6	Data with Ack Path List Size	SWORD	0 = Dynamique > 0 = Nombre max de membres
7	Data with Ack Path List	STRUCT of	
		SWORD	Nombre de membres dans la matrice
		ARRAY of	
		UINT	Connection Instance ID (identificateur d'instance de connexion)
		USINT	Longueur de chemin
		Padded EPATH	Chemin

4.1.8.5.2 Services spécifiques à un objet

4.1.8.5.2.1 Add_AckData_Path

Le corps de l'Add_AckData_Path_RequestPDU doit être tel que spécifié dans le Tableau 86.

Tableau 86 – Structure du corps de l'Add_AckData_Path_RequestPDU

Nom	Type de donnée
Connection Instance ID (identificateur d'instance de connexion)	UINT
Consumer Path Size	USINT
Consumer Path	Padded EPATH

Le corps de l'Add_AckData_Path_ResponsePDU est vide.

4.1.8.5.2.2 Remove_AckData_Path

Le corps de l'Add_AckData_Path_RequestPDU doit être tel que spécifié dans le Tableau 87.

Tableau 87 – Structure du corps de Remove_AckData_Path_RequestPDU

Nom	Type de donnée
Connection Instance ID (identificateur d'instance de connexion)	UINT

Le corps de Remove_AckData_Path_RequestPDU est vide.

4.1.8.6 Éléments de syntaxe spécifiques à un objet Time Sync

4.1.8.6.1 Attributs

Attributs d'instance

Le format des attributs d'instance d'objet Time Sync doit être tel que spécifié dans le Tableau 88.

Tableau 88 – Attributs d'instance d'objet Time Sync

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	PTPEnable	BOOL	PTP activé pour ce dispositif: 0 = PTP non activé 1 = PTP activé
2	IsSynchronized	BOOL	Horloge locale synchronisée avec l'horloge de référence: 0 = horloge locale non synchronisée 1 = horloge locale synchronisée
3	SystemTimeMicroseconds	ULINT	Valeur courante de local_time en microsecondes
4	SystemTimeNanoseconds	ULINT	Valeur courante de local_time en nanosecondes
5	OffsetFromMaster	LINT	Décalage entre l'horloge locale et l'horloge-mère en nanosecondes
6	MaxOffsetFromMaster	ULINT	Décalage maximal entre l'horloge locale et l'horloge-mère en nanosecondes
7	MeanPathDelayToMaster	LINT	Retard de chemin par rapport à la mère
8	GrandMasterClockInfo	STRUCT of	Informations d'horloge grand-mère
	ClockIdentity	USINT[8]	Identificateur unique de l'horloge. Voir Tableau 89 pour le codage de ClockIdentity.
	ClockClass	UINT	Classe de la qualité d'horloge. Le Tableau 90 montre les valeurs les plus susceptibles d'être utilisées par l'objet Time sync. Voir la CEI 61588:2009 pour une liste complète de valeurs.
	TimeAccuracy	UINT	Exactitude absolue prévue de l'horloge par rapport à l'époque PTP. Voir Tableau 91 pour les valeurs de TimeAccuracy.
	OffsetScaledLogVariance	UINT	Mesure des propriétés intrinsèques de stabilité de l'horloge
	CurrentUtcOffset	UINT	Décalage TUC courant, en secondes, par rapport au Temps atomique international (TAI) de l'horloge. Au 1er janvier 2006 à 0 h TUC, le décalage était de 33 secondes
	TimePropertyFlags	WORD	Pour la valeur de TimePropertyFlags, voir le Tableau 92.

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
	TimeSource	UINT	Pour les valeurs possibles de TimeSource, voir le Tableau 93.
	Priority1	UINT	Priorité relative de l'horloge grand-mère par rapport aux autres horloges dans le système. Voir l'attribut 16 Priority1
	Priority2	UINT	Priorité relative de l'horloge grand-mère par rapport aux autres horloges dans le système. Voir l'attribut 17 Priority2
9	ParentClockInfo	STRUCT of	Informations d'horloge parente
	ClockIdentity	USINT[8]	Identificateur unique de l'horloge. Voir Tableau 89 pour le codage de ClockIdentity.
	ClockClass	UINT	Classe de la qualité d'horloge. Le Tableau 90 montre les valeurs les plus susceptibles d'être utilisées par l'objet Time sync. Voir la CEI 61588:2009 pour une liste complète de valeurs.
	Time Accuracy	UINT	Exactitude absolue prévue de l'horloge par rapport à l'époque PTP. Voir Tableau 91 pour les valeurs de TimeAccuracy.
	OffsetScaledLogVariance	UINT	Mesure des propriétés intrinsèques de stabilité de l'horloge
	CurrentUtcOffset	UINT	Décalage TCU courant, en secondes, par rapport au Temps atomique international (TAI) de l'horloge. Au 1er janvier 2006 à 0 h TUC, le décalage était de 33 secondes
	TimePropertyFlags	WORD	Pour la valeur de TimePropertyFlags, voir le Tableau 92.
	TimeSource	UINT	Pour les valeurs possibles de TimeSource, voir le Tableau 93.
	Priority1	UINT	Priorité relative de l'horloge grand-mère par rapport aux autres horloges dans le système. Voir l'attribut 16 Priority1
	Priority2	UINT	Priorité relative de l'horloge grand-mère par rapport aux autres horloges dans le système. Voir l'attribut 17 Priority2
10	LocalClockInfo	STRUCT of	Informations d'horloge locale
	ClockIdentity	USINT[8]	Identificateur unique de l'horloge Voir le Tableau 89 pour le codage de ClockIdentity.
	ClockClass	UINT	Classe de la qualité d'horloge. Le Tableau 90 montre les valeurs les plus susceptibles d'être utilisées par l'objet Time sync. Voir la CEI 61588:2009 pour une liste complète de valeurs.
	Time Accuracy	UINT	Exactitude absolue prévue de l'horloge par rapport à l'époque PTP. Voir Tableau 91 pour les valeurs de TimeAccuracy.

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
	OffsetScaledLogVariance	UINT	Mesure des propriétés intrinsèques de stabilité de l'horloge
	CurrentUtcOffset	UINT	Décalage TCU courant, en secondes, par rapport au Temps atomique international (TAI) de l'horloge. Au 1er janvier 2006 à 0 h TUC, le décalage était de 33 secondes
	TimeSource	UINT	Pour les valeurs possibles de TimeSource, voir le Tableau 93.
	Priority1	UINT	Priorité relative de l'horloge grand-mère par rapport aux autres horloges dans le système. Voir l'attribut 16 priority1
11	NumberOfPorts	INT	Priorité relative de l'horloge grand-mère par rapport aux autres horloges dans le système. Voir l'attribut 17 priority2
12	PortStateInfo	STRUCT of	Informations d'état de port
	NumberOfPorts	UINT	Nombre de ports PTP sur le dispositif
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port
	PortState	UINT	L'état courant du port PTP: 1 = Initializing (s'initialise) 2 = Faulty (défectueux) 3 = Disabled (désactivé) 4 = Listening (à l'écoute) 5 = Pre_Master 6 = Master (maître) 7 = Passive (passif) 8 = Uncalibrated (non étalonné) 9 = Slave (esclave) All other (tout autre) = Reserved (réservé)
13	PortEnableCfg	STRUCT of	Configuration d'activation de port
	NumberOfPorts	UINT	Nombre de ports PTP sur le dispositif
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port
	PortEnable	UINT	PortEnable a les valeurs suivantes (la valeur par défaut est "activé"): 1 = port activé 0 = port désactivé
14	PortLogAnnounceIntervalCfg	STRUCT of	Configuration de l'intervalle entre annonces de journal de port
	NumberOfPorts	UINT	Nombre de ports PTP sur le dispositif
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port
	PortLogAnnounceInterval	UINT	Intervalle d'annonces PTP entre des messages "Announce" successifs émis par l'horloge-mère
15	PortSyncLogIntervalCfg	STRUCT of	Configuration de l'intervalle entre annonces de journal de sync de port
	NumberOfPorts	UINT	Nombre de ports PTP sur le dispositif
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
	PortSyncLogInterval	UINT	Intervalle de sync PTP entre des messages "Sync" successifs émis par l'horloge-mère
16	Priority1	USINT	Cet attribut spécifie la notation Best Master de cette horloge et annule et remplace la qualité d'horloge (classe, exactitude et variance).
17	Priority2	USINT	Cet attribut spécifie la notation "Best Master" de cette horloge après que la qualité d'horloge (classe, exactitude et variance) a été évaluée et annule et remplace le tie-breaker .
18	DomainNumber	USINT	Domaine d'horloge PTP
19	ClockType	WORD	Pour les valeurs de ClockType, voir le Tableau 94
20	ManufactureIdentity	USINT[4]	Identité de fabricant de l'horloge. Les 3 premiers octets spécifient l'OUI (Organization Unique ID «identificateur propre à une organisation») de l'IEEE pour le fabricant. Le dernier octet est réservé.
21	ProductDescription	STRUCT of	
	Size	UDINT	Taille de la description du produit
	Description	USINT[Size]	Description de produit du dispositif qui contient l'horloge
22	RevisionData	STRUCT of	
	Size	UDINT	Taille de la date de révision
	Revision	USINT[Size]	Données de révision du dispositif qui contient l'horloge
23	UserDescription	STRUCT of	
	Size	UDINT	Taille de la description de l'utilisateur
	Description	USINT[Size]	UserDescription spécifie la description d'utilisateur du dispositif qui contient l'horloge.
24	PortProfileIdentityInfo	STRUCT of	Le profil de port doit être: 00-21-6C-00-01-00
	NumberOfPorts	UINT	Nombre de ports PTP sur le dispositif
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port
	PortProfileIdentity	USINT[8]	Identité de profil de port
25	PortPhysicalAddressInfo	STRUCT of	Informations d'adresse physique de port
	NumberOfPorts	UINT	Nombre de ports PTP sur le dispositif
		ARRAY of STRUCT	
	PortNumber	UINT	Numéro d'identité du port
	PhysicalProtocol	UINT	Protocole physique
	SizeOfAddress	UINT	Taille de PortPhysicalAddress
	PortPhysicalAddress	USINT[16]	Adresse physique de port
26	PortProtocolAddressInfo	STRUCT of	Informations d'adresse de protocole de port
	NumberOfPorts	UINT	Nombre de ports
		ARRAY of STRUCT	
	PortNumber	UINT	Le numéro de l'identité de port

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
	NetworkProtocol	UINT	Valeurs de protocole de réseau 1 = UDP/IPv4 (par exemple: CP 2/2) 2 = UDP/IPv6 3 = ISO/IEC 8802-3 4 = CP 2/3 5 = CP 2/1 0xFFFF = protocole local ou inconnu tout autre = réservé
	SizeOfAddress	UINT	Taille de PortProtocolAddress
	PortProtocolAddress	USINT[16]	Adresse de protocole de port
27	StepRemoved	UINT	Nombre de chemins de communication traversés entre l'horloge locale et l'horloge grand-mère
28	SytemTimeAndOffset	STRUCT of	Temps système et décalage
	SystemTime	ULINT	Temps système en microsecondes
	SystemOffset	ULINT	Décalage de la valeur d'horloge locale

Tableau 89 – Codage de ClockIdentity pour différentes mises en œuvre de réseau

Réseau	Octet 0	Octet 1	Octet 2	Octet 3	Octet 4	Octet 5	Octet 6	Octet 7
CP 2/1	0xFF	0x02	ID de vendeur		Numéro de série			
CP 2/2	Adresse MAC			0xFF	0xFE	Adresse MAC		
CP 2/3	0xFF	0x01	ID de vendeur		Numéro de série			
Local ou fermé	0xFF	0xFF	Vendor ID		Numero de série			
Tous les autres	Voir CEI 61588:2009.							

Tableau 90 – Valeurs de ClockClass

ClockClass	Valeur
Référence primaire	6
Référence primaire (Hold)	7
Référence dégradée A (Maître seulement)	52
Référence dégradée B (Maître / Esclave)	187
Valeur par défaut	248
Esclave seulement	255
Voir CEI 61588:2009.	tous les autres

Tableau 91 – Valeurs de TimeAccuracy

Exactitude d'horloge	Valeur
Réservé	0x00 à 0x1F
± 25 ns	0x20
± 100 ns	0x21
± 250 ns	0x22
± 1 µs	0x23
± 2,5 µs	0x24
± 10 µs	0x25
± 25 µs	0x26
± 100 µs	0x27
± 250 µs	0x28
± 1 ms	0x29
± 2,5 ms	0x2A
± 10 ms	0x2B
± 25 ms	0x2C
± 100 ms	0x2D
± 250 ms	0x2E
± 1 s	0x2F
± 10 s	0x30
> ± 10 s	0x31
Reserved	0x32-0x7F
Pour utilisation par des profils PTP alternatifs	0x80-0xFD
Inconnu	0xFE
Réservé	0xFF

Tableau 92 – Valeurs de bit de TimePropertyFlags

TimePropertyFlags	Indice de bit
Leap indicator 61	0
Leap indicator 59	1
Décalage TUC courant valide	2
Échelle de temps PTP	3
Temps identifiable	4
Fréquence identifiable	5

Tableau 93 – Valeurs de TimeSource

TimeSource	Valeur
ATOMIC CLOCK	0x10
GPS	0x20
TERRESTRIAL RADIO	0x30
PTP	0x40
NTP	0x50
HAND SET	0x60
OTHER	0x90
INTERNAL OSCILLATOR	0xA0
Pour utilisation par des profils PTP alternatifs	0xF0 à 0xFE
Réservé	0xFF

Tableau 94 – Types de Clock

Type de Clock	Indice de bit
Horloge ordinaire	0
Horloge frontière	1
Horloge transparente pour les homologues	2
Horloge transparente de bout en bout	3
Nœud de gestion	4
Réservé	Tous les autres bits

4.1.8.6.2 Services spécifiques à un objet

Il n'y a pas de services spécifiques à un objet.

4.1.8.7 Éléments de syntaxe spécifiques à un objet Parameter

4.1.8.7.1 Attributs

4.1.8.7.1.1 Attributs de classe

Le format des attributs de classe d'objets Parameter doit être tel que spécifié dans le Tableau 84.

Tableau 95 – Attributs de classe d'objets Parameter

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
2	Max Instance	WORD	Le numéro d'instance le plus élevé d'un objet créé à ce niveau de hiérarchie de classe
8	Parameter Class Descriptor	UINT	Voir Tableau 96

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
9	Configuration Assembly Instance	UINT	Cet attribut doit être mis à zéro si l'assemblage de configuration n'est pas pris en charge.
10	Native Language	UINT	0 = Anglais 1 = Français 2 = Espagnol 3 = Italien 4 = Allemande 5 = Japon 6 = Portugais 7 = Chinois mandarin

L'attribut Parameter Class Descriptor (descripteur de classe de paramètre) contient les bits pour décrire des caractéristiques de paramètre. Les bits pris en charge sont spécifiés dans le Tableau 96.

Tableau 96 – Valeurs des bits de Parameter Class Descriptor

Bit	Définition
0	Prend en charge les instances de Parameter. 1 = les instances individuelles de Parameter SONT prises en charge. 0 = AUCUNE instance individuelle de Parameter n'est prise en charge. Seule instance d'assemblage de configuration utilisée
1	Prend en charge les attributs complets (Full) 1 = Tous les attributs complets de Parameter SONT pris en charge. 0 = Seuls les attributs réduits d'instance de Parameter sont pris en charge.
2	Doit faire la commande sauvegarde de stockage non volatil 1 = Doit exécuter la commande de sauvegarde de stockage non volatil. Le service Save de la classe de paramètre est utilisé pour sauvegarder des valeurs de paramètre d'instance. 0 = Peut ne pas exécuter la commande sauvegarde de stockage non volatil.
3	Les paramètres sont stockés dans le stockage non volatil. 1 = Tous les paramètres complets sont stockés dans un stockage non volatil 0 = Des paramètres ne sont pas stockés dans un stockage non volatil

4.1.8.7.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Parameter est tel que spécifié dans le Tableau 85.

Tableau 97 – Attributs d'instance d'objet Parameter

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Parameter Value	Data type (Type de données) spécifié dans Descriptor, Data Type et Data Size	
2	Link Path Size	USINT	Nombre d'octets
3	Link Path	Packed EPATH	Le Link Path (chemin de liaison) est limité à 255 octets
4	Descriptor	WORD	Voir Tableau 98
5	Data Type	EPATH	
6	Data Size	USINT	
7	Parameter Name String	SHORT_STRING	Le nombre maximum de caractères est 16.
8	Units String	SHORT_STRING	Le nombre maximum de caractères est 4.
9	Help String	SHORT_STRING	Le nombre maximum de caractères est 64.
10	Minimum Value	Data Type	Voir Tableau 99
11	Maximum Value	Data Type	Voir Tableau 99
12	Default Value	Data Type	Valeur effective à laquelle il convient de régler le paramètre lorsque l'utilisateur souhaite la valeur par défaut pour le paramètre
13	Scaling Multiplier	UINT	Voir Tableau 100
14	Scaling Divisor	UINT	Voir le Tableau 100
15	Scaling Base	UINT	Voir le Tableau 100
16	Scaling Offset	INT	Voir le Tableau 100
17	Multiplier Link	UINT	Voir le Tableau 101
18	Divisor Link	UINT	Voir le Tableau 101
19	Base Link	UINT	Voir le Tableau 101
20	Offset Link	UINT	Voir le Tableau 101
21	Decimal Precision	USINT	Spécifie le nombre de décimales à utiliser pour afficher la valeur d'étude mise à l'échelle. Utilisé également pour déterminer la valeur d'incrément effective faisant que le fait d'incrémenter une valeur entraîne un changement de la valeur d'étude mise à l'échelle avec cette précision.
22	International Parameter Name	STRINGI	Chaîne lisible par l'homme qui représente le nom du paramètre
23	International Engineering Units	STRINGI	Unités d'étude associées au paramètre
24	International Help String	STRINGI	Chaîne lisible par l'homme qui représente la chaîne d'aide

Tableau 98 – Sémantique de l'attribut d'instance Descriptor

Bit	Définition	Signification
0	Prend en charge le chemin réglable.	Indique que le chemin de liaison peut être établi.
1	Prend en charge les chaînes énumérées.	Indique que les chaînes énumérées sont prises en charge et peuvent être lues avec le service Get_Enum_String.
2	Prend en charge la mise à l'échelle.	Indique qu'il convient que le facteur d'échelle soit implémenté de manière à présenter la valeur à l'utilisateur en unités d'étude.
3	Prend en charge les liaisons de mise à l'échelle	Indique que les valeurs pour le facteur d'échelle peuvent être récupérées d'autres paramètres. Si le bit descripteur "Supports Scaling" («prend en charge la mise à l'échelle») est également mis et la valeur de liaison de mise à l'échelle est zéro, la valeur d'échelle doit être utilisée. Si le bit descripteur "Supports Scaling" («prend en charge la mise à l'échelle») est également mis et la valeur de liaison de mise à l'échelle n'est pas zéro, la valeur de liaison de mise à l'échelle doit être utilisée.
4	Paramètre en lecture seule	Indique que l'attribut Parameter Value peut seulement être lu, et pas mis.
5	Paramètre de surveillance	Indique que l'attribut Parameter Value est mis à jour en temps réel par le dispositif.
6	Prend en charge la mise à l'échelle de précision étendue.	Indique que le facteur d'échelle de précision étendue doit être implémenté de manière à présenter la valeur à l'utilisateur en unités d'étude.
7	Prend en charge des chaînes énumérées non consécutives.	Désuet
8	Autorise l'énumération et aussi les valeurs individuelles.	Désuet
9	Paramètre non affiché	L'outil de configuration ne doit pas afficher ce paramètre.
10	Référence de paramètre indirecte	La valeur du paramètre représente le numéro d'instance d'un autre paramètre. L'instance du paramètre référencé ne doit pas être une référence indirecte supplémentaire. Une valeur de paramètre égale à 0 doit indiquer l'absence de référence. Le type de données pour ce paramètre doit être USINT, UINT ou UDINT.
11	Non adressable	Le paramètre n'est adressable directement à partir du réseau par aucun des services et_Attribute or Get_Attribute.
12	"Save" (sauvegarder) pris en charge	Ce bit, lorsqu'il est mis, indique que ce paramètre peut être individuellement sauvegardé en envoyant le service Save à cette instance de paramètre.
13	"Apply" (appliquer) pris en charge	Ce bit, lorsqu'il est mis, indique que l'exécution du service Apply sur cette instance de paramètre est requise avant que les changements d'attribut n'altèrent le comportement de l'instance.
14	Paramètre en écriture seule	Indique que l'attribut Parameter Value peut seulement être mis, et pas lu.
15	Réservé	Il doit être mis à zéro.

Tableau 99 – Sémantique de Minimum Value et de Maximum Value

Type de donnée	Description et sémantique	Sémantique de Minimum Value	Sémantique de Maximum Value	Requis/Facultatif/Interdit
OCTET	Bit String – longueur de 8 bits	Le bit de poids fort qui est mis dans la valeur minimale est le bit valide de poids le plus faible devant être énuméré. Le bit de poids fort qui est mis dans la valeur maximale est le bit valide de poids le plus fort devant être énuméré. Une valeur minimale égale à zéro signifie que le bit 0 est le bit valide de poids le plus faible devant être énuméré. Une valeur maximale égale à zéro n'est pas valide. Exemple 1: Valeur min = 0b00001 Valeur max = 0b01000 Les bits 0 à 3 sont énumérés. Exemple 2: Valeur min = 0b00001 Valeur max = 0b01011 Les bits 0 à 3 sont énumérés, les bits 0 et 1 dans valeur max sont ignorés. Exemple 3: Valeur min = 0b00000 Valeur max = 0b01000 Les bits 0 à 3 sont énumérés, la valeur min 0 sous-entend le bit 0.		Facultatif
WORD	Bit String – longueur de 16 bits			
DWORD	Bit String – longueur de 32 bits			
LWORD	Bit String – longueur de 64 bits			
STRING ^a	String (indicateur de longueur 2 octets, 1 octet par caractère)	Longueur de chaîne minimale	Longueur de chaîne maximale	Requis
STRING2 ^a	String (indicateur de longueur 2 octets, 2 octets par caractère)	Longueur de chaîne minimale	Longueur de chaîne maximale	Requis
STRINGN ^a	String (indicateur de longueur 2 octets, N octets par caractère)	Longueur de chaîne minimale	Longueur de chaîne maximale	Requis
SHORT_STRING ^a	String (indicateur de longueur 1 octet, caractères de 1 octet)	Longueur de chaîne minimale	Longueur de chaîne maximale	Requis
STRINGI ^a	Chaîne de caractères internationaux	Longueur minimale (en caractères) du composant "stringcontents" de la séquence implicite de la chaîne	Longueur maximale (en caractères) du composant "stringcontents" de la séquence implicite de la chaîne	Requis
EPATH ^a	Path (chemin) codé	Longueur de chaîne minimale	Longueur de chaîne maximale	Facultative
ENGUNIT	Unités d'étude	La valeur minimale pour ce type de données n'est pas définie et ne doit pas être spécifiée dans fichier EDS ou l'instance d'objet paramètre.	La valeur maximale pour ce type de données n'est pas définie et ne doit pas être spécifiée dans fichier EDS ou l'instance d'objet paramètre.	Facultatif
Tous les autres types de données		La valeur numérique minimale qui peut être attribuée à la valeur de donnée	La valeur numérique maximale qui peut être attribuée à la valeur de donnée	Facultatif ^b
^a Les types de données STRING, STRING2, STRINGN, SHORT_STRING, STRINGI et EPATH n'ont pas de spécification de valeur minimale ou maximale. Les champs de valeur minimale et maximale sont utilisés pour présenter les longueurs maximale et minimale de chemin ou de chaîne. Dans ces cas, le paramètre Data Size est utilisé pour représenter le nombre d'octets requis par caractère ou entrée de codage. ^b S'il n'est pas spécifié de Minimum Value et/ou de Maximum Value, la valeur minimale et/ou maximale pour la valeur de donnée de paramètre est/sont telle(s) que définie(s) dans la CEI 61158-5-2:2010.				

Tableau 100 – Attributs de formule de mise à l'échelle

Attribut	Signification
Multiplier Value	(Mult) Valeur de multiplicateur pour la formule de mise à l'échelle. Cette valeur peut être basée sur un autre paramètre.
Divisor Value	Div) Valeur de diviseur pour la formule de mise à l'échelle. Cette valeur peut être basée sur un autre paramètre spécifié dans le Divisor Link.
Base Value	(Base) Valeur de base pour la formule de mise à l'échelle. Cette valeur peut être basée sur un autre paramètre spécifié dans le Base Link.
Offset Value	(Offset) Valeur de décalage pour la formule de mise à l'échelle. Cette valeur peut être basée sur un autre paramètre spécifié dans l'Offset Link. Cet attribut est un INT et peut être négatif.
Decimal Precision	(Precision) Valeur de précision dans la formule de mise à l'échelle. Cette valeur ne peut pas être basée sur un autre paramètre.

Liaisons de mise à l'échelle

Les valeurs de mise à l'échelle (multiplicateur, diviseur, base et décalage) peuvent aussi être basées sur d'autres paramètres. Les liaisons de mise à l'échelle spécifient de quelle instance la valeur de mise à l'échelle en question doit être récupérée. Si l'attribut de mise à l'échelle est mis à 0, cette valeur de mise à l'échelle est une constante et n'est liée à aucun autre paramètre. Le Tableau 101 spécifie les liaisons de mise à l'échelle.

Tableau 101 – Liaisons de mise à l'échelle

Attribut	Signification
Multiplier Link	Spécifie l'instance de paramètre d'où la valeur de multiplicateur (Multiplier Value) est récupérée.
Divisor Link	Spécifie l'instance de paramètre d'où la valeur de diviseur (Divisor Value) est récupérée.
Base Link	Spécifie l'instance de paramètre d'où la valeur de base (Base Value) est récupérée.
Offset Link	Spécifie l'instance de paramètre d'où la valeur de décalage (Offset Value) est récupérée.

Attributs de facteur d'échelle

Le facteur d'échelle représente les valeurs de paramètre UINT et INT effectives dans d'autres formats. La formule (1) doit être utilisée pour déterminer la valeur d'étude à partir de la valeur effective.

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) \times \text{Mult} \times \text{Base}}{\text{Div}} \quad (1)$$

La valeur d'étude peut ensuite être affichée à l'utilisateur dans les termes spécifiés dans l'attribut d'instance Decimal Precision. L'inverse de la formule de mise à l'échelle doit être utilisé pour déterminer la valeur effective à partir de la valeur d'étude (voir la Formule (2)).

$$\text{ActualValue} = \frac{(\text{EngValue} \times \text{Div})}{\text{Mult} \times \text{Base}} - \text{Offset} \quad (2)$$

Le facteur d'échelle de précision étendue ajoute de la précision étendue au facteur d'échelle normalisé. La formule (3) doit être utilisée pour déterminer la valeur d'étude à partir de la valeur effective.

$$\text{EngValue} = \frac{(\text{ActualValue} + \text{Offset}) \times \text{Mult} \times \text{Base}}{\text{Div} \times 10^{\text{Precision}}} \quad (3)$$

La valeur d'étude peut ensuite être affichée à l'utilisateur dans les termes spécifiés dans l'attribut Decimal Precision. L'inverse de la formule de mise à l'échelle de précision étendue doit être utilisé pour déterminer la valeur effective à partir de la valeur d'étude (voir la Formule (4)).

$$\text{ActualValue} = \frac{(\text{EngValue} \times \text{Div} \times 10^{\text{Precision}})}{\text{Mult} \times \text{Base}} - \text{Offset} \quad (4)$$

4.1.8.7.2 Service spécifique à un objet

Get_Enum_String

Le corps de Get_Enum_String_RequestPDU doit être tel que spécifié dans le Tableau 102.

Tableau 102 – Structure du corps de Get_Enum_String_RequestPDU

Nom	Type de donnée	Sémantique des valeurs
Enumerated String Number	USINT	Nombre de chaînes énumérées à récupérer. Plage de valeurs maximale possible = 0 à 255. La valeur indique le décalage de bits (par rapport à zéro) pour les énumérations de bits, la valeur effective (par rapport à zéro) pour les énumérations de valeurs.

Le corps de Get_Enum_String_ResponsePDU doit être tel que spécifié dans le Tableau 103.

Tableau 103 – Structure du corps de Get_Enum_String_ResponsePDU

Nom	Type de donnée	Sémantique des valeurs
Enumerated String	SHORT_STRING	Chaînes énumérées Nombre maximum de caractères = 16

Les chaînes énumérées sont des chaînes lisibles par l'homme qui décrivent soit un bit, soit une valeur, cela dépendant du type de données du paramètre. La relation du type de chaîne au type de données de paramètre est montrée dans le Tableau 104.

Tableau 104 – Type de chaîne énumérée versus type de données de paramètres

Si le type de données du paramètre est:	les chaînes énumérées retournées sont:
OCTET, WORD, DWORD, LWORD	Chaînes énumérées de bits
USINT, SINT, UINT, INT, BOOL	Chaînes énumérées de valeurs
Tous les autres types de données	Non valide pour l'énumération

Si le type de données du paramètre est OCTET, WORD, DWORD ou LWORD, le fait de demander un service Get_Enum_String avec un numéro de chaîne énumérée de 0 sur un paramètre retourne un SHORT_STRING qui décrit le bit 0. Le fait de demander un service Get_Enum_String avec un numéro de chaîne énumérée de 1 sur un paramètre retourne un SHORT_STRING qui décrit le bit 1, et ainsi de suite jusqu'au numéro de bit maximal pris en charge par le paramètre.

Si le type de données du paramètre est SINT, USINT, INT, UINT ou BOOL, le fait de demander un service Get_Enum_String avec un numéro de chaîne énumérée de 0 sur un paramètre retourne un SHORT_STRING qui décrit la valeur égale à 0. Le fait de demander un service Get_Enum_String avec un numéro de chaîne énumérée de 1 sur un paramètre retourne un SHORT_STRING qui décrit la valeur égale à 1.

4.1.8.8 Éléments de syntaxe spécifique à un Connection Manager

4.1.8.8.1 Attributs

4.1.8.8.1.1 Attributs de classe

Le format des attributs de classe d'objets Connection Manager doit être tel que spécifié dans le Tableau 105.

Tableau 105 – Attributs de classe d'objets Connection Manager

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
2	Max Instance	UINT	
4	Optional Attribute List	STRUCT of	
	Number attributes	UINT	
	Optional attributes	ARRAY of UINT	

4.1.8.8.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Connection Manager est tel que spécifié dans le Tableau 106.

Tableau 106 – Attributs d'instance d'objet Connection Manager

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	OpenReqs	UINT	
2	OpenFormat Rejects	UINT	
3	OpenResource Rejects	UINT	
4	OpenOther Rejects	UINT	
5	CloseReqs	UINT	
6	CloseFormat Rejects	UINT	
7	CloseOther Rejects	UINT	
8	ConnTimeouts	UINT	Le nombre total de temporisations de connexion qui se sont produites dans les connexions contrôlées par ce Connection Manager.
9	Connection Entry List	STRUCT of	

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
	NumConnEntries	UINT	Nombre de bits utilisés dans l'élément ConnOpenBits Cet attribut, divisé par 8 et incrémenté pour tout reste éventuel, donne la longueur en octets de la matrice dans le champ ConnOpenBits.
	ConnOpenBits	ARRAY of BOOL	Champ de bits. Liste de données de connexion. Chaque bit représente une connexion possible. 0= Aucune connexion 1 = Connexion établie
11	CpuUtilization	UINT	Utilisation de l'UCT en dixièmes de pourcent. Plage de 0 à 1 000 représentant 0 % à 100 %.
12	MaxBuffSize	UDINT	La quantité d'espace tampon est en octets.
13	BufSize Remaining	UDINT	La quantité d'espace tampon est en octets.

4.1.8.9 Éléments de syntaxe spécifiques à un objet Connexion

4.1.8.9.1 Attributs

4.1.8.9.1.1 Attributs de classe

Le format des attributs de classe d'objets Connection doit être tel que spécifié dans le Tableau 107.

Tableau 107 – Attributs de classe d'objets Connection

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Revision	UINT	Révision de cet objet. Valeur = 1
2	Max Instance	UINT	

4.1.8.9.1.2 Attributs d'instance

Le format des attributs d'instance d'objet Connection est tel que spécifié dans le Tableau 108.

Tableau 108 – Attributs d'instance d'objet Connection

ID d'attr	Nom	Type de Données	Sémantique
1	État	USINT	État de l'objet
2	Instance_type	USINT	Indique soit E/S, soit Messaging Connection (connexion de messagerie).
3	TransportClass_trigger	SWORD	Définit le comportement de la connexion
4	CP2/3_produced_connection_id	UINT	Placé dans le CAN Identifier Field (champ identificateur de CAN) de ISO 11898 lorsque la connexion émet sur un sous-réseau CP 2/3.
5	CP2/3_consumed_connection_id	UINT	La valeur de CAN Identifier Field de l'ISO 11898 qui désigne le message à recevoir sur un sous-réseau CP 2/3.
6	CP2/3_initial_comm_characteristics	SWORD	Définit le(s) Message Group à travers le(s)quel(s) les productions et les consommations associées à cette connexion ont lieu sur un sous-réseau CP 2/3.
7	Produced_connection_size	UINT	Nombre maximum d'octets émis à travers cette connexion.
8	Consumed_connection_size	UINT	Nombre maximum d'octets reçus à travers cette connexion.
9	Expected_packet_rate	UINT	Définit la synchronisation associée à cette connexion
10	CFP2_produced_connection_id	UDINT	Identifie le message envoyé sur le sous-réseau par cette connexion.
11	CPF2_consumed_connection_id	UDINT	Identifie le message reçu en provenance du sous-réseau pour cette connexion.
12	Watchdog_timeout_action	USINT	Définit comment gérer les temporisations d'inactivité/chien de garde.
13	Produced_connection_path_length	UINT	Nombre d'octets dans l'attribut produced_connection_path.
14	Produced_connection_path	Packed EPATH	Spécifie l'/les objet(s) d'application dont les données doivent être produites par cet objet Connection. Voir 4.1.9.
15	Consumed_connection_path_length	UINT	Nombre d'octets dans l'attribut consumed_connection_path.
16	Consumed_connection_path	Packed EPATH	Spécifie l'/les objet(s) d'application qui doivent recevoir les données consommées par cet objet Connection. Voir 4.1.9.
17	Production_inhibit_time	UINT	Définit le temps minimum entre productions de nouvelles données. Cet attribut est requis pour toutes les connexions Client E/S, excepté celles ayant un déclencheur de production "Cyclic".
18	Connection_timeout_multiplier	USINT	Spécifie le multiplicateur appliqué à la valeur expected_packet_rate pour dériver la valeur du temporisateur d'inactivité/chien de garde (Inactivity/Watchdog Timer).

ID d'attr	Nom	Type de Données	Sémantique
19	Connection_binding_list	STRUCT UINT Array of UINT	Liste des instances de connexion E/S liées à cette instance.

State

Cet attribut définit l'état courant de l'instance de Connection. Le Tableau 109 définit les états possibles et attribue une valeur utilisée pour indiquer l'état en question.

Tableau 109 – Valeurs attribuées à l'attribut d'état

Valeur	Nom d'état	Description
00	Non-existant (inexistant)	La connexion n'a pas encore été instanciée.
01	Configuring (en configuration)	La connexion a été instanciée et attend que les événements suivants se produisent: (1) être configurée correctement et (2) être invitée à appliquer la configuration.
02	Waiting For Connection ID ^a (en attente de l'ID de connexion)	L'instance de Connection attend exclusivement que soient fixées les valeurs de son/ses attribut(s) consumed_connection_id et/ou produced_connection_id. ^b
03	Established (établie)	La connexion a été configurée de façon valide/complète et la configuration a été appliquée avec succès.
04	Timed Out (temporisée)	Si un objet Connection subit une temporisation d'activité/chien de garde, une transition <u>peut</u> être effectuée vers cet état. Pour plus de détails, voir la description de l'attribut watchdog_timeout_action et celle de Inactivity/Watchdog.
05	Deferred Delete ^a (suppression différée)	Si un objet Explicit Messaging Connection (connexion de messagerie explicite) subit une temporisation d'activité/chien de garde, une transition <u>peut</u> être effectuée vers cet état. Pour plus de détails, voir la description de l'attribut watchdog_timeout_action et celle de Inactivity/Watchdog Timer.
06	Closing (en cours de fermeture)	Un objet Bridged Connection (connexion pontée) a reçu, et traite, un service Forward_Close provenant du Connection Manager. La suppression de la connexion ne se produira pas tant qu'une réponse Forward_Open réussie n'a pas été reçue du nœud-cible.
^a Cette valeur est seulement utilisée sur CP 2/3. ^b Lorsque les attributs d'instance Connection sont appliqués, il peut ne pas être possible pour le module de générer les valeurs de produced_connection_id et/ou de consumed_connection_id (voir l'attribut initial_comm_characteristics pour les règles). Si tel est le cas, toutes les tâches requises pour appliquer les attributs sont exécutées à l'exception de l'initialisation de ces attributs dans l'objet Connection et les objets Producteur/Consommateur de liaison associés et une transaction est effectuée vers cet état. Dans cet état, l'instance de Connection attend exclusivement que soient fixées les valeurs de son/ses attribut(s) produced_connection_id et/ou consumed_connection_id.		

Important: Une instance de connexion créée de manière dynamique est l'enfant de la connexion de messagerie explicite à travers laquelle elle a été créée. Des types de sous-réseau différents peuvent fournir d'autres mécanismes pour créer une connexion qui impliquent une relation parent-enfant particulière. Pour des détails, voir les spécifications spécifiques à un réseau.

Important: Toutes les ressources associées à une Connection Instance (A) qui a été créée de manière dynamique à travers une Explicit Messaging Connection (B) doivent être libérées si l'Explicit Messaging Connection (B) expire avant que la Connection Instance créée de manière dynamique (A) ne passe à l'état Established.

Important: Lorsqu'une transition est effectuée vers l'état Established, tous les temporisateurs associés à l'objet Connection sont activés (voir Connection Timing («temporisation de connexion») dans la CEI 61158-5-2:2010, paragraphe 6.2.3.2.1.7)

Instance_type

Cet attribut définit le type d'instance (voir Tableau 110).

Tableau 110 – Valeurs attribuées à l'attribut instance_type

Valeur	Signification
00	Explicit Messaging (messagerie explicite). Cette instance de Connection représente l'un des points d'extrémité d'une Explicit Messaging Connection. Une Explicit Messaging Connection est créée de manière dynamique en envoyant l' <i>Open Explicit Messaging Connection Request</i> (demande d'ouvrir une connexion de messagerie explicite) à la Connection Class (classe de connexion).
01	I/O (E/S). Cette instance de Connection représente l'un des points d'extrémité d'une I/O Connection (connexion E/S). Une I/O Connection est créée de manière dynamique en envoyant une <i>Create Request</i> (demande "Créer") à la Connection Class.
02	Bridged (pontée). Cette instance de Connection représente un saut intermédiaire d'une connexion I/O ou Explicit Messaging pontée. Une paire d'objets Bridged Connection (un pour chaque sous-réseau ponté entre eux) est créée de manière dynamique par un nœud après avoir reçu avec succès un service Forward_Open vers l'objet Connection Manager lorsque ce nœud n'est pas le point d'extrémité (la cible).

TransportClass_trigger

Définit s'il s'agit d'une connexion productrice seulement, consommatrice seulement, ou à la fois productrice et consommatrice. Si ce point d'extrémité doit effectuer une production de données, cet attribut définit aussi l'événement qui déclenche la production. Les huit (8) bits sont divisés comme montré à la Figure 7.

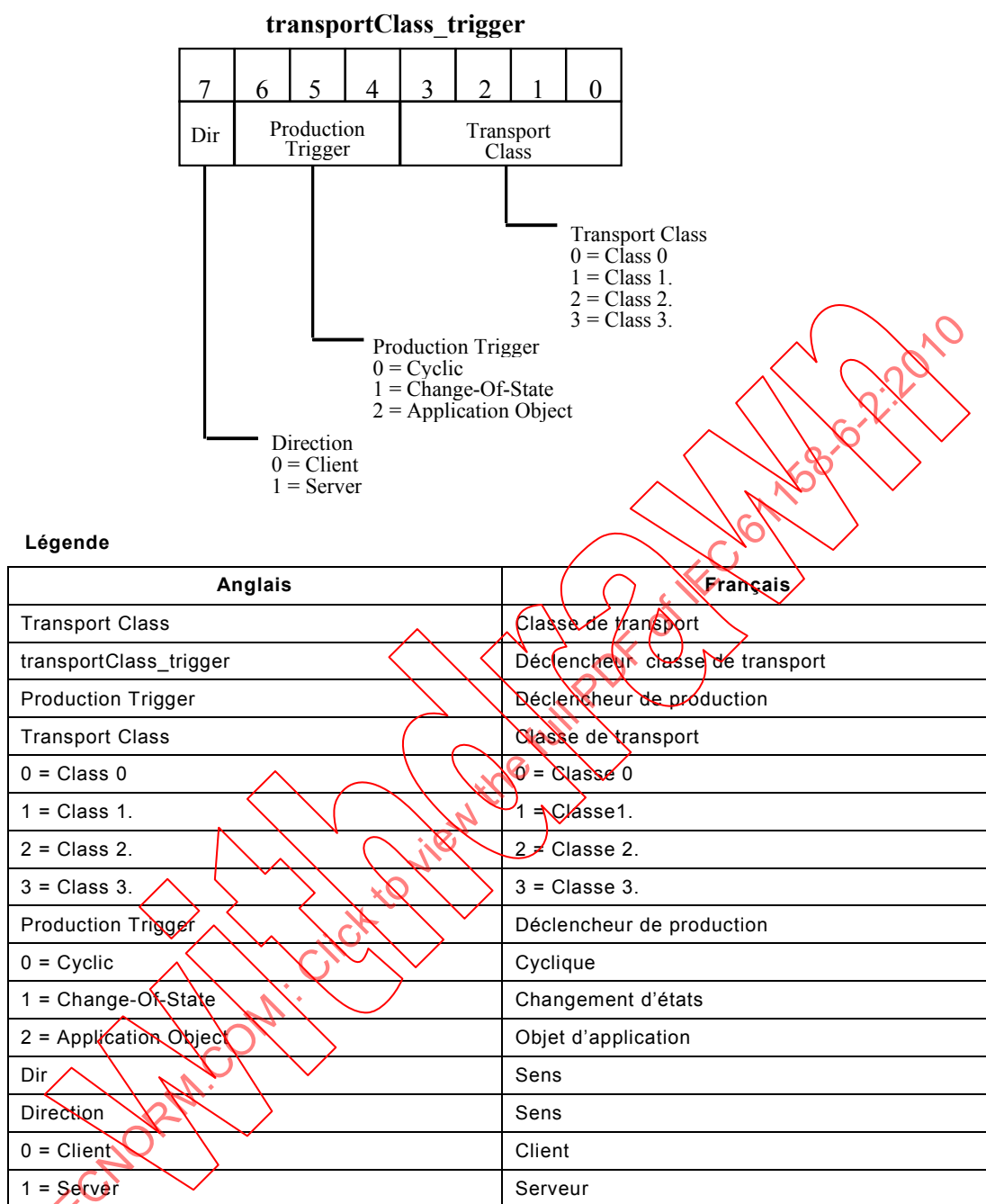


Figure 7 – AttributTransport Class_Trigger

Le **bit Direction** de l'octet transportClass_trigger indique si le point d'extrémité doit agir comme Client ou comme Serveur sur cette connexion (voir Tableau 111).

Tableau 111 – Valeurs possibles dans le bit Direction

Valeur	Signification	
0	Client	Ce point d'extrémité donne le comportement en Client associé à cette connexion. En outre, cette valeur indique que les bits <i>Production Trigger</i> dans l'octet transportClass_trigger contiennent la description de l'instant où le Client doit produire le message associé à cette connexion. Les connexions clientes avec la valeur du déclencheur de production de 0 ou 1 (Cyclic ou Change-of-State) doivent produire immédiatement après la transition vers l'état Established.
1	Server; Serveur	Ce point d'extrémité donne le comportement en Serveur associé à cette connexion. En outre, cette valeur indique que les bits <i>Production Trigger</i> dans l'octet transportClass_trigger doivent être IGNORED (ignorés). Les bits <i>Production Trigger</i> sont ignorés en raison du fait qu'un point d'extrémité Serveur réagit à l'émission à partir du Client. Le seul moyen par lequel un point d'extrémité Serveur est déclenché pour émettre est lorsque cette réaction appelle la production d'un message (classes de transport 2 ou 3).

Le Tableau 112 énumère les valeurs qui sont possibles dans les bits **Production Trigger** de l'attribut **transportClass_trigger**.

Tableau 112 – Valeurs possibles dans les bits Production Trigger

Si la valeur est:	La production d'un message est:	
0	Cyclic	L'expiration du temporisateur Transmission Trigger Timer déclenche la production de données. Pour une description détaillée du Transmission Trigger Timer, voir la CEI 61158-5-2:2010, paragraphe 6.2.3.2.1.7 Connection Timing (temporisation de connexion).
1	ChangeOf-State	La production a lieu lorsqu'un changement d'état est détecté par l'objet d'application. NOTE Le point d'extrémité consommateur peut avoir été configuré pour attendre le paquet à un certain débit, quel que soit le mécanisme de déclenchement au point d'extrémité producteur. Pour plus d'informations, voir la description de l'attribut expected_packet_rate d'un objet Connection et la description de Connection Timing dans la CEI 61158-5-2:2010, paragraphe 6.2.3.2.1.7.
2	Application Object Triggered	L'objet d'application décide du moment auquel déclencher la production. NOTE Le point d'extrémité consommateur peut avoir été configuré pour attendre le paquet à un certain débit, quel que soit le mécanisme de déclenchement au point d'extrémité producteur. Pour plus d'informations, voir la description de l'attribut expected_packet_rate d'un objet Connection et la description de Connection Timing dans la CEI 61158-5-2:2010, paragraphe 6.2.3.2.1.7.
3	7	Reserved (réservés)

Le Tableau 113 énumère les valeurs possibles dans le quartet **Transport Class** de l'attribut **transportClass_trigger**. Les comportements résultant de ces valeurs particulières sont illustrés dans la série de figures qui suivent le tableau.

Tableau 113 – Valeurs possibles dans les bits Transport Class

Valeur	Signification	
0	Transport Class 0	En fonction de la valeur du bit <i>Dir</i> , ce point d'extrémité de connexion sera un point d'extrémité producteur seulement OU consommateur seulement. À la suite de l'application de cette instance de Connexion, le module instancie soit un Link Producer (bit <i>Dir</i> = Client, producteur seulement), soit un Link Consumer (bit <i>Dir</i> = Server, consommateur seulement) devant être associé à cette connexion.
1	Transport Class 1	
2	Transport Class 2	Indique que le module produira ET consommera à travers cette connexion. Le point d'extrémité Client génère la première production de données qui est consommée par le Server, amenant le Server à retourner une production qui est consommée par le Client.
3	Transport Class 3	
4	Transport Class 4	Non bloquant.
5	Transport Class 5	Non bloquant, fragmentant.
6	Transport Class 6	Multicast (multidiffusion), fragmentant.
7 - F	Reserved (réservés)	

Une valeur de compte de séquences de 16 bits est ajoutée en préfixe au début de tous les transports de classes 1, 2 et 3. Cette valeur est utilisée pour détecter la livraison de doublons de paquets de données. Les valeurs de compte de séquences sont initialisées sur la première production de messages et incrémentées sur chaque nouvelle production ultérieure de données. Un renvoi d'anciennes données ne doit pas changer le compte de séquences et un consommateur doit ignorer les données lorsqu'elles sont reçues avec un doublon de compte de séquences. Les applications consommatrices peuvent utiliser ce mécanisme pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion.

Cependant, sur CP 2/3, les classes de transport 2 et 3 n'ajoutent pas en préfixe un compte de séquences de 16 bits comme décrit en 4.1.4.

Les tableaux et figures suivants illustrent les combinaisons valides de **Production Trigger** et de **Transport Class** et donnent une description des comportements de Client et de Server. Voir la CEI 61158-5-2:2010, paragraphe 6.2.3.2.1.7, pour une description du *Transmission Trigger Timer*, qui est montré dans les illustrations.

En résumé, se référer au Tableau 114 pour les valeurs valides dans l'attribut **transportClass_trigger** d'une instance de connexion

Tableau 114 – Attribut TransportClass_Trigger

Bits TransportClass_trigger	Signification
1 xxx 0000	Direction = Server, Production Trigger = IGNORED, Transport Class = 0.
1 xxx 0001	Direction = Server, Production Trigger = IGNORED, Transport Class = 1.
1 xxx 0010	Direction = Server, Production Trigger = IGNORED, Transport Class = 2.
1 xxx 0011	Direction = Server, Production Trigger = IGNORED, Transport Class = 3. Il s'agit de la valeur attribuée à cet attribut au sein du point d'extrémité Server d'une Explicit Messaging Connection.
0 000 0000	Direction = Client, Production Trigger = Cyclic, Transport Class = 0.
0 000 0001	Direction = Client, Production Trigger = Cyclic, Transport Class = 1.
0 000 0010	Direction = Client, Production Trigger = Cyclic, Transport Class = 2.
0 000 0011	Direction = Client, Production Trigger = Cyclic, Transport Class = 3.
0 001 0000	Direction = Client, Production Trigger = ChangeOfState, Transport Class = 0.
0 001 0001	Direction = Client, Production Trigger = ChangeOfState, Transport Class = 1.
0 001 0010	Direction = Client, Production Trigger = ChangeOfState, Transport Class = 2.
0 001 0011	Direction = Client, Production Trigger = ChangeOfState, Transport Class = 3.
0 010 0000	Direction = Client, Production Trigger = Application object, Transport Class = 0.
0 010 00001	Direction = Client, Production Trigger = Application object, Transport Class = 1.
0 010 0010	Direction = Client, Production Trigger = Application object, Transport Class = 2.
0 010 0011	Direction = Client, Production Trigger = Application object, Transport Class = 3. Il s'agit de la valeur attribuée à cet attribut au sein du point d'extrémité Client d'une Explicit Messaging Connection.
1 111 1111	Valeur par défaut attribuée à cet attribut dans une connexion E/S.

CP2/3_produced_connection_id,

Cet attribut est requis pour un sous-réseau CP 2/3. Les autres types de sous-réseau ne doivent pas utiliser cet attribut.

Il contient l'ID de connexion CP 2/3 à associer aux émissions envoyées sur cette connexion (le cas échéant), à savoir, la valeur qui sera spécifiée dans le CAN Identifier Field de l'ISO 11898 lorsque cette Connection émet. Voir la CEI 62026-3:2008, 5.1.2 pour une description de la manière dont CP 2/3 utilise le champ CAN Identifier Field. Cette valeur est chargée directement dans l'attribut associé *connection_id* du Link Producer. Les valeurs sont définies dans le Tableau 115.

Tableau 115 – Valeurs définies pour l'attribut CP2/3_produced_connection_id

Valeur	Signification
0 7F0 _{hex}	La valeur à placer dans le CAN Identifier Field lorsque cette connexion émet.
800 _{hex} à FFFE _{hex}	Réservée
FFFF _{hex}	Valeur par défaut attribuée à cet attribut dans une connexion E/S. Cet attribut retiendra cette valeur si cette instance de Connection ne produit pas de donnée (consommateur seulement).

CP2/3_consumed_connection_id

Cet attribut est requis pour un sous-réseau CP 2/3. Les autres types de sous-réseau ne doivent pas utiliser cet attribut.

Il contient l'ID de connexion CP qui identifie les messages devant être reçus sur cette connexion (le cas échéant), à savoir, la valeur de CAN Identifier Field qui est associée aux messages que cet objet Connection reçoit. Voir la CEI 62026-3:2008, 5.1.2 pour une description de la manière dont CP 2/3 utilise le champ CAN Identifier Field. Cette valeur est chargée directement dans l'attribut associé *connection_id* du Link Consumer. Les valeurs sont définies dans le Tableau 116.

Tableau 116 – Valeurs définies pour l'attribut CP2/3_consumed_connection_id

Valeur	Signification
0 7F0 _{hex}	La valeur qui identifie les messages à consommer. Elle sera spécifiée dans le CAN Identifier Field des messages qui sont à consommer.
800 _{hex} à FFFE _{hex}	Reserved
FFFF _{hex}	Valeur par défaut attribuée à cet attribut dans une connexion E/S. Cet attribut retiendra cette valeur si cette instance de Connection ne consomme pas de donnée (producteur seulement).

CP2/3_initial_comm_characteristics

Cet attribut est requis pour un sous-réseau CP 2/3. Les autres types de sous-réseau ne doivent pas utiliser cet attribut.

Il définit le(s) Message Group à travers le(s)quel(s) les productions et les consommations associées à cette connexion ont lieu.

Cet octet est divisé en deux quartets, comme montré à la Figure 8.

7	6	5	4	3	2	1	0
Initial Production Characteristics				Initial Consumption Characteristics			

Légende

Anglais	Français
Initial Production Characteristics	Caractéristiques de production initiales
Initial Consumption Characteristics	Caractéristiques de consommation initiales

Figure 8 – Format de l'attribut CP2/3_initial_comm_characteristics

Le Tableau 117 énumère les valeurs qui sont possibles au sein du quartet (quartet supérieur) Initial Production Characteristics («Caractéristiques de production initiales») de l'attribut CP2/3_initial_comm_characteristics.

Tableau 117 – Valeurs pour le quartet Initial Production Characteristics

Valeur	Signification	
0	Produire à travers le Message Group 1	La production associée à cette connexion a lieu à travers le Message Group 1. Le module producteur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_produced_connection_id de l'objet Connection. Le module producteur alloue un ID de message issu de son groupement d'ID de message de Groupe 1 et le combine avec son ID de MAC Source pour générer l'ID de connexion. L'ID de message de Groupe 1 ayant la valeur numérique la plus faible est à utiliser pour générer la valeur de l'attribut CP2/3_produced_connection_id. Cette valeur doit aussi être chargée dans l'/les attribut(s) CP2/3_consumed_connection_id associé(s) à l'objet ou aux objets Connection consommateur(s).
1	Produire à travers le Message Group 2 (Destination)	La production associée à cette connexion a lieu à travers le Message Group 2. En outre, l'ID de MAC (Destination MAC ID) du destinataire prévu est à placer au sein du composant ID de MAC du Field Identifiant de groupe 2. Dans ce cas, le module consommateur génère la valeur de l'ID de connexion à associer aux émissions sur cette connexion. Lorsque le module consommateur a généré cette valeur et l'a chargée dans l'attribut approprié CP2/3_consumed_connection_id de l'objet Connection, elle peut être lue et ensuite chargée dans l'attribut CP2/3_produced_connection_id de l'objet Connection producteur.
2	Produire à travers le Message Group 2 (Source)	La production associée à cette connexion a lieu à travers le Message Group 2. En outre, l'ID de MAC (Source MAC ID) du module producteur est à placer au sein du composant ID de MAC du Field Identifiant de groupe 2. Dans ce cas, le module producteur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_produced_connection_id de l'objet Connection. L'ID de message de Groupe 2 ayant la valeur numérique la plus faible est à utiliser pour générer la valeur de l'attribut CP2/3_produced_connection_id. Cette valeur doit aussi être chargée dans l'/les attribut(s) CP2/3_consumed_connection_id associé(s) à l'objet ou aux objets Connection consommateur(s).
3	Produire à travers le Message Group 3	La production associée à cette connexion a lieu à travers le Message Group 3. Le module producteur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_produced_connection_id de l'objet Connection. Le module producteur alloue un ID de message issu de son groupement d'ID de message de Groupe 3 et le combine avec son ID de MAC Source pour générer l'ID de connexion. L'ID de message de Groupe 3 ayant la valeur numérique la plus faible est à utiliser pour générer la valeur de l'attribut CP2/3_produced_connection_id. Cette valeur doit aussi être chargée dans l'/les attribut(s) CP2/3_consumed_connection_id associé(s) à l'objet ou aux objets Connection consommateur(s).
4 à 0xE	Réserve	
0xF	Valeur par défaut	La valeur par défaut attribuée au quartet Initial Production Characteristics («Caractéristiques de production initiales») de CP2/3 au sein de la connexion E/S. NOTE S'il s'agit d'une connexion E/S seulement consommatrice, la valeur par défaut reste dans ce quartet. Les objets Explicit Messaging Connection configurent automatiquement cet attribut lorsque la connexion est établie.

Le Tableau 118 énumère les valeurs possibles au sein du quartet (quartet inférieur) Initial Consumption Characteristics («Caractéristiques de consommation initiales») de l'attribut CP2/3_initial_comm_characteristics.

Tableau 118 – Valeurs pour le quartet Initial Consumption Characteristics

Valeur	Signification	
0	Consommer un message de groupe 1	Le message à consommer sera émis à travers le Message Group 1. Le module producteur génère la valeur d'ID de connexion. Cette valeur doit être chargée dans l'attribut CP2/3_consumed_connection_id associé à l'objet ou aux objets Connection consommateur(s).
1	Consommer un message de groupe 2 (Destination)	Le message à consommer sera émis à travers le Message Group 2. L'ID de MAC (Destination MAC ID) du destinataire prévu est spécifié au sein de l'identificateur de groupe 2. Le module consommateur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_consumed_connection_id associé à cet objet Connection. L'ID de message de Groupe 2 ayant la valeur numérique la plus faible est à utiliser pour générer la valeur de l'attribut CP2/3_consumed_connection_id. Cette valeur doit être chargée dans l'attribut CP2/3_produced_connection_id de l'objet Connection producteur.
2	Consommer un message de groupe 2 (Source)	Le message à consommer sera émis à travers le Message Group 2. L'ID de MAC (Source MAC ID) du module émetteur est spécifié au sein de l'identificateur de groupe 2. Dans ce cas, le module producteur génère la valeur d'ID de connexion et la charge dans l'attribut CP2/3_produced_connection_id de l'objet Connection. Cette valeur doit être chargée dans l'attribut CP2/3_consumed_connection_id associé à l'objet ou aux objets Connection consommateur(s).
3	Consommer un message de groupe 3	Le message à consommer sera émis comme un message de groupe 3. Le module producteur génère la valeur d'ID de connexion. La valeur d'ID de connexion doit être chargée cet attribut CP2/3_consumed_connection_id de l'objet Connection.
4 à 0xE	Reserved	
0xF	Valeur par défaut	La valeur par défaut attribuée au quartet Initial Consumption Characteristics («Caractéristiques de consommation initiales») de CP 2/3 au sein de la connexion E/S. NOTE – S'il s'agit d'une connexion E/S seulement productrice, la valeur par défaut reste dans ce quartet. Les Explicit Messaging Connection configurent automatiquement cet attribut lorsque la connexion est établie.

Important: Le module qui génère un ID de connexion doit garantir qu'il n'alloue pas la paire ID de message/ID de MAC de sorte que deux modules distincts soient capables d'émettre des profils binaires identiques au sein de l'Identifier Field.

Produced_connection_size

La signification de cet attribut est différente pour les Explicit Messaging Connection par rapport à celle qu'elle est pour les I/O Connection. Si le sous-réseau définit un protocole de fragmentation et le dispositif prend en charge la fragmentation, cette taille peut être supérieure à la plus grande taille de trame. Pour plus de détails, voir les spécifications d'adaptation spécifiques à un réseau.

Pour les Explicit Messaging Connections:

Cet attribut signifie le nombre maximal d'octets de données de demande/réponse de routeur de messages (Message Router Request/Response Data) (voir 4.1.8) qu'un dispositif est capable d'émettre sur cette connexion. Les dispositifs qui imposent une limite connue à la quantité maximale de données de demande/réponse de routeur de messages qui peuvent être émises dans un seul message, ou une seule série fragmentée, initialisent cet attribut en conséquence. Les dispositifs qui ne peuvent pas ou ne prédefinisent pas une limite d'émission préalable placent la valeur 0xffff dans cet attribut (il peut encore exister une limite, mais elle n'est pas connue à l'avance).

Important: En raison de la nature de la messagerie explicite (Explicit Messaging), la longueur des messages explicites variera pendant la durée de vie d'une connexion. Les Explicit Messaging Connection accomplissent la fragmentation en fonction de la *longueur* du message courant à émettre lorsque le type de sous-réseau peut définir un protocole de fragmentation.

Pour les I/O Connection:

Si le **transportClass_trigger** indique que cette instance de Connection doit produire, cet attribut définit la quantité maximale de données E/S qui peuvent être produites comme un *seul bloc* sur cette connexion. La quantité de données E/S à transmettre à un instant donné quelconque peut être inférieure ou égale à l'attribut **connection_size**.

Cet attribut a zéro (0) comme valeur par défaut au sein de la connexion E/S. Si le type de sous-réseau prend en charge la fragmentation et cet attribut est mis à une valeur supérieure à la plus grande charge utile dans une I/O Connection, la connexion morcellera les données en plusieurs fragments. Pour plus de détails, voir les spécifications d'adaptation spécifiques à un réseau.

Important: La fragmentation au sein des connexions E/S est accomplie en fonction de la valeur dans cet attribut, quelle que soit la quantité courante de données à émettre.

Important: Les messages E/S qui ne contiennent pas de données E/S d'application et étaient configurés pour contenir des données (par le fait que `produced_connection_size` est supérieur à zéro (0)) sont définis pour indiquer un événement *No Data* (aucune donnée) pour l'objet ou les objets d'application de réception. Le comportement d'un objet d'application à la détection d'un événement *No Data* est spécifique à chaque objet d'application.

Consumed_connection_size

La signification de cet attribut est différente pour les Explicit Messaging Connection par rapport à celle qu'elle est pour les I/O Connection. Si le sous-réseau définit un protocole de fragmentation et le dispositif prend en charge la fragmentation, cette taille peut être supérieure à la plus grande taille de trame. Pour plus de détails, voir les spécifications d'adaptation spécifiques à un réseau.

Pour les Explicit Messaging Connections:

Cet attribut signifie le nombre maximal d'octets de données de demande/réponse de routeur de messages (Message Router Request/Response Data) qu'un dispositif est capable de recevoir sur cette connexion.

Les dispositifs qui imposent une limite connue à la quantité maximale de données de demande/réponse de routeur de messages qui peuvent être reçues dans un seul message, ou une seule série fragmentée, initialisent cet attribut en conséquence. Les dispositifs qui ne peuvent pas ou ne prédéfinissent pas une limite de réception préalable placent la valeur 0xffff dans cet attribut (il peut encore exister une limite, mais elle n'est pas connue à l'avance). En raison de la nature de la messagerie explicite (Explicit Messaging), la longueur des messages explicites variera pendant la durée de vie d'une connexion.

Pour les I/O Connection:

Si l'attribut **transportClass_trigger** indique que cette connexion doit consommer, cet attribut définit la quantité maximale de données qui peuvent être reçues comme un *seul bloc* sur cette connexion. La quantité réelle de données E/S reçues à un instant donné quelconque peut être inférieure ou égale à l'attribut **connection_size**.

Cet attribut a zéro (0) comme valeur par défaut au sein de la connexion E/S. Si le type de sous-réseau prend en charge la fragmentation et cet attribut est mis à une valeur supérieure à la plus grande charge utile dans une I/O Connection, la connexion traitera le protocole de fragmentation. Pour plus de détails, voir les spécifications d'adaptation spécifiques à un réseau.

La longueur d'un message E/S doit être inférieure ou égale à son attribut pour qu'un objet I/O Connection le reçoive comme étant un message valide. Si un objet I/O Connection reçoit un message dont la longueur est supérieure à cet attribut, il rejette immédiatement le message et interrompt tout traitement ultérieur.

NOTE En ce qui concerne le point d'extrémité Server d'une connexion de classe de transport 2 ou 3, cette condition d'erreur *trop de données* se traduit en ce qu'aucune réponse n'est émise et le temporisateur chien de garde n'est pas éjecté (kicked).

Expected_packet_rate

Cet attribut est utilisé pour générer les valeurs chargées dans le *Transmission Trigger Timer* et l'*Inactivity/Watchdog Timer*. Pour une description détaillée des temporisateurs *Transmission Trigger* et *Inactivity/Watchdog*, voir la CEI 61158-5-2:2010, paragraphe 6.2.3.2.1.7.

La résolution de cet attribut est en millisecondes. Une demande de configurer cet attribut peut se traduire en la spécification d'une valeur temporelle à laquelle un produit ne peut pas satisfaire. En plus de l'accomplissement d'une vérification de plage spécifique à un produit lorsqu'une demande de modifier cet attribut est reçue, les étapes suivantes sont accomplies:

- Si la valeur spécifiée n'est pas égale à un incrément de la résolution d'horloge disponible, la valeur est arrondie **par excès** à la valeur pratique suivante. Par exemple: Une demande *Set_Attribute_Single* reçue spécifie la valeur 5 pour l'attribut **expected_packet_rate** et le produit assure une résolution de 10 millisecondes sur les temporisateurs. Dans ce cas, le produit chargerait la valeur 10 dans l'attribut **expected_packet_rate**.
- La valeur qui est effectivement chargée dans l'attribut **expected_packet_rate** est rapportée dans le *Service Data Field* (champ de données de service) d'un message de réponse *Set_Attribute_Single* associé à une demande de modifier cet attribut.
- Si la valeur demandée est égale à un incrément de la résolution d'horloge, la valeur demandée est chargée dans l'attribut **expected_packet_rate** et rapportée dans la réponse. Par exemple: Si la valeur 100 est demandée et la résolution d'horloge est 10 millisecondes, une valeur de 100 est chargée.
- Lorsqu'un objet *Connection* est dans l'état **Established**, toutes les éventuelles modifications de l'attribut **expected_packet_rate** ont un effet immédiat sur le temporisateur *Inactivity/Watchdog Timer*. Les étapes suivantes sont accomplies par un objet *Connection* dans l'état **Established** lorsqu'une demande est reçue pour modifier l'attribut **expected_packet_rate**:
 - le temporisateur courant *Inactivity/Watchdog Timer* est annulé,
 - un nouvel *Inactivity/Watchdog Timer* est activé en fonction de la nouvelle valeur contenue dans l'attribut **expected_packet_rate**.

Cet attribut a 2500 (2 500 ms) comme valeur par défaut au sein des *Explicit Messaging Connections* et zéro (0) comme valeur par défaut au sein d'une *I/O Connection*.

CPF2_produced_connection_id

Contient l'ID de connexion, qui identifie les messages à envoyer sur cette connexion (le cas échéant). Cet attribut ne doit pas être mis en œuvre lorsque le type de sous-réseau est CP 2/3.

CPF2_consumed_connection_id

Contient l'ID de connexion, qui identifie les messages à recevoir sur cette connexion (le cas échéant). Cet attribut ne doit pas être mis en œuvre lorsque le type de sous-réseau est CP 2/3.

Watchdog_timeout_action

Cet attribut définit l'action qu'il convient que l'objet Connection accomplisse lorsque l'Inactivity/Watchdog Timer expire. Le Tableau 119 définit les spécificités de cet attribut.

Tableau 119 – Valeurs de watchdog_timeout_action

Valeur	Signification
0	<i>Passer à Timed Out (temporisé)</i> . La connexion passe à l'état Timed Out (temporisé) et reste dans cet état jusqu'à ce qu'elle soit Reset (réinitialisée) ou Deleted (supprimée). La commande de Reset ou Delete pourrait provenir d'une source interne (par exemple, un objet application) ou du réseau (par exemple: un outil de configuration). Il s'agit de la valeur par défaut de cet attribut en ce qui concerne les I/O Connection. Cette valeur est non valide pour les Explicit Messaging Connection.
1	<i>Auto Delete (autosuppression)</i> . La classe de connexion supprime automatiquement la connexion si elle subit une temporisation d'inactivité/chien de garde (Inactivity/Watchdog timeout). Il s'agit de la valeur par défaut de cet attribut en ce qui concerne les Explicit Messaging Connection.
2	<i>Auto Reset (autoréinitialisation)</i> . La connexion reste dans l'état Established et redémarre immédiatement le temporisateur Inactivity/Watchdog timer. Cette valeur est non valide pour les Explicit Messaging Connection.
3	<i>Deferred Delete (suppression différée)</i> . La connexion passe à l'état Deferred (différé) si des instances de connexion enfant quelconques sont dans l'état Established. Si aucune instance de connexion enfant n'est dans l'état Established, la connexion est supprimée. Cette valeur est seulement utilisée sur CP 2/3 et est non valide pour les I/O Messaging Connection.
4 FF	Reserved

Produced_connection_path_length

Spécifie le nombre d'octets d'information au sein de l'attribut **produced_connection_path**. Il est automatiquement initialisé lorsque l'attribut **produced_connection_path** est configuré. Cet attribut a zéro (0) comme valeur par défaut.

Produced_connection_path

L'attribut **produced_connection_path** est constitué d'un train d'octets qui définit l'/les objet(s) d'application dont les données sont à produire par cet objet Connection. **Le format de ce train d'octets est spécifié en 4.1.9.** Comme valeur par défaut, cet attribut est vide à la suite de l'instanciation de la connexion. Il reste vide au sein des Explicit Messaging Connection et au sein des objets Connection qui ne produisent pas.

Consumed_connection_path_length

Spécifie le nombre d'octets d'information au sein de l'attribut **consumed_connection_path**. Il est automatiquement initialisé lorsque l'attribut **consumed_connection_path** est configuré. Cet attribut a zéro (0) comme valeur par défaut.

Consumed_connection_path

L'attribut **consumed_connection_path** est constitué d'un train d'octets qui définit l'/les objet(s) d'application qui doivent recevoir les données consommées par cet objet Connection. **Le format de ce train d'octets est spécifié en 4.1.9.** Comme valeur par défaut, cet attribut est vide à la suite de l'instanciation de la connexion. Il reste vide au sein des Explicit Messaging Connection et au sein des objets Connection qui ne consomment pas.

Production_inhibit_time

Cet attribut est utilisé pour configurer le retard minimum entre nouvelles productions de données. Il est requis pour toutes les connexions Client E/S, excepté celles ayant un déclencheur de production "Cyclic". Le service Set_Attribute_Single doit être pris en charge lorsque cet attribut est mis en œuvre. Une valeur de zéro (la valeur par défaut pour cet attribut) indique l'absence de temps de neutralisation.

La résolution de cet attribut est en millisecondes. Une demande de configurer cet attribut peut se traduire en la spécification d'une valeur temporelle à laquelle un produit ne peut pas satisfaire. En plus de l'accomplissement d'une vérification de plage spécifique à un produit lorsqu'une demande de modifier cet attribut est reçue, les étapes suivantes sont accomplies:

- Si la valeur spécifiée n'est pas égale à un incrément de la résolution d'horloge disponible, la valeur est arrondie **par excès** à la valeur pratique suivante. Par exemple: Une demande Set_Attribute_Single reçue spécifie la valeur 5 pour l'attribut **production_inhibit_time** et le produit assure une résolution de 10 millisecondes sur les temps. Dans ce cas, le produit chargerait la valeur 10 dans l'attribut **production_inhibit_time**.
- La valeur qui est effectivement chargée dans l'attribut **production_inhibit_time** est rapportée dans le Service Data Field (champ de données de service) d'un message de réponse Set_Attribute_Single associé à une demande de modifier cet attribut.
- Si la valeur demandée est égale à un incrément de la résolution d'horloge, la valeur demandée est chargée dans l'attribut **production_inhibit_time** et rapportée dans la réponse. Par exemple: la valeur 100 est demandée et la résolution d'horloge est 10 millisecondes.

La valeur de **production_inhibit_time** est chargée dans le temporisateur Production Inhibit Timer chaque fois qu'une nouvelle production de données a lieu.

Lorsqu'un objet Connection est dans l'état **Established**, toutes les éventuelles modifications de l'attribut **production_inhibit_time** n'ont aucun effet sur le Production Inhibit Timer fonctionnant actuellement. La nouvelle valeur de production_inhibit_time est chargée dans le temporisateur Production Inhibit Timer sur la nouvelle production suivante de données..

Lorsque le service apply_attributes est reçu, le **production_inhibit_time** doit être vérifié par rapport à l'attribut **expected_packet_rate**. Si la valeur de **expected_packet_rate** est supérieure à zéro, mais inférieure à la valeur de **production_inhibit_time**, une erreur doit être retournée. Dans ce cas, lorsque deux valeurs d'attribut sont en conflit, utiliser l'ID de l'attribut **production_inhibit_time** comme étant le code d'erreur supplémentaire envoyé dans la réponse d'erreur.

Connection_timeout_multiplier

Le Connection_timeout_multiplier spécifie le multiplicateur appliqué au débit de paquet prévu pour obtenir la valeur utilisée par le temporisateur Inactivity/Watchdog Timer. Pour les valeurs énumérées de cet attribut, voir la description du paramètre Connection Timeout Multiplier au sein de Paramètres de service spécifiques à un objet Connection Manager . La valeur par défaut pour cet attribut est zéro (spécifiant un multiplicateur de 4).

Connection_binding_list

L'attribut Connection_binding_list identifie les instances de connexion qui sont liées à cette connexion. La structure de l'attribut fournit un compte 16 bits de connexions liées, suivi d'une liste d'instances liées. Cet attribut doit être pris en charge si le service Connection_Bind est pris en charge.

4.1.8.9.2 Services spécifiques à un objet

4.1.8.9.2.1 Service Connection_Bind

Le service spécifique à un objet Connection_Bind lie ensemble deux instances de I/O connexion créées de manière dynamique à des fins de temporisations et de suppressions de connexion. Une I/O connexion créée de manière dynamique est le résultat d'un service Create à la classe Connection avec l'attribut Instance_type mis à I/O (valeur d'attribut de 1). Si l'une de ces instances de I/O temporise ou est supprimée, l'autre doit manifester le même comportement.

Le corps de Connection_Bind_RequestPDU doit être tel que défini dans le Tableau 120.

Tableau 120 – Structure du corps de Connection_Bind_RequestPDU

Nom de paramètre	Type de Données	Description de paramètre
Bound Instances	STRUCT of UINT UINT	Nombres d'instances des objets Connection à lier

Le corps de Connection_Bind_ResponsePDU est vide.

La réponse de service peut retourner un statut issu de la liste de codes de statut dans le Tableau 121.

Tableau 121 – Statut spécifique à un objet pour le service Connection_Bind

Code de statut général	Code de statut étendu	Description de statut
0x02	0x01	L'une ou les deux instances de connexion sont inexistantes.
0x02	0x02	La classe et/ou l'instance de connexion est/sont à court de ressources pour lier des instances.
0x0C	0x01	Toutes les deux instances de connexion existent, mais l'une au moins n'est pas dans l'état Established.
0x20	0x01	Toutes les deux instances de connexion sont la même valeur.
0xD0	0x01	L'une ou les deux instances de connexion n'est pas/ne sont pas une connexion E/S créée de manière dynamique.
0xD0	0x02	L'une ou les deux instances de connexion a/ont été créée(s) en interne et le dispositif n'autorise pas une liaison à lui.

4.1.8.9.2.2 Service Producing_Application_Lookup

Le service spécifique à un objet Producing_Application_Lookup fournit un mécanisme pour trouver une ou plusieurs instances de connexion dans l'état Established produisant des données à partir d'un objet d'application donné. Si le chemin d'application productrice requis est en train d'être produit par une connexion quelconque dans l'état Established, le nombre d'instances de chaque connexion produisant à partir de ce chemin est retourné.

Le corps de Producing_Application_Lookup_RequestPDU doit être tel que spécifié dans le Tableau 122.

Tableau 122 – Structure du corps de Producing_Application_Lookup_RequestPDU

Nom de paramètre	Type de Données	Description de paramètre
Producing Application Path	EPATH	Le chemin de connexion de l'application productrice à rechercher au sein des connexions dans l'état Established. Le chemin doit être un simple Logical EPATH ou Symbolic EPATH.

Le corps de Producing_Application_Lookup_ResponsePDU doit être tel que spécifié dans le Tableau 123.

Tableau 123 – Structure du corps de Producing_Application_Lookup_ResponsePDU

Nom de paramètre	Type de Données	Description de paramètre
Instance Count	UINT	Nombre d'instances retourné dans le paramètre Connection Instance List.
Connection Instance List	Struct of UINT	Liste de nombres d'instances de connexion produisant les données à partir du chemin de connexion demandé au sein du nœud.

La réponse de service peut retourner un statut issu de la liste de codes de statut dans le Tableau 124.

Tableau 124 – Codes de statut du service Producing_Application_Lookup

Code de statut général	Code de statut étendu	Description de statut
0x02	0x01	Le chemin de connexion n'a été trouvé dans aucune instance dans l'état Established.

4.1.8.9.2.3 Service SafetyOpen

Voir la CEI 61784-3-2 pour la description de ce service.

4.1.8.9.2.4 Service SafetyClose

Voir la CEI 61784-3-2 pour la description de ce service.

4.1.9 Chemins de message et de connexion

4.1.9.1 Contenu

Le chemin doit spécifier l'élément d'objet qui est soit la cible d'une demande de connexion, soit la destination d'une demande de message. Le chemin doit contenir plusieurs segments qui peuvent indiquer l'itinéraire jusqu'au prochain nœud (dans le cas de plusieurs liaisons), à quoi se connecter ou quand envoyer un message au dispositif-cible. Chaque segment doit être constitué d'un octet de descripteur de segment qui spécifie le type de segment et les informations de segment dont la taille dépend du type de segment. Les deux types de chemin doivent être

- les chemins bourrés;
- les chemins condensés.

Les deux formats de chemin (bourré et condensé) ne doivent pas être interchangeables. L'utilisation de bourré versus condensé doit dépendre du contexte d'utilisation. Si aucun format n'est spécifié de façon explicite pour un chemin, c'est le format bourré qui doit être utilisé.

Chaque segment d'un chemin bourré doit être un mot de 16 bits aligné. Si un octet de bourrage est requis pour obtenir l'alignement, le segment doit spécifier l'emplacement de l'octet de bourrage. Un chemin condensé ne doit pas contenir d'octets de bourrage.

Les types de segment possibles doivent être comme suit:

- Port segment (segment de port) (où);
- Logical segment (segment logique) (quoi);
- Network segment (segment de réseau) (quand ou comment);
- Symbolic segment (segment symbolique);
- Data segment (segment de données).

4.1.9.2 Type de segment

Chaque segment du chemin doit contenir un type de segment pour indiquer comment interpréter le segment. Le segment est contenu dans le premier octet du segment. Les bits du premier octet d'un segment de chemin doivent être numérotés de 0 à 7, où le bit 0 doit être le bit de poids faible de l'octet. Les bits du type de segment sont définis à la Figure 9.

Segment Type Depends on segment Type



0 0 0	Port
0 0 1	Logical
0 1 0	Network
0 1 1	Symbolic
1 0 0	Data / Extended symbolic
1 0 1	Reserved
1 1 0	Reserved
1 1 1	Reserved

Légende

Anglais	Français
Segment Type	Type de segment
Depends on segment Type	Dépend du type de segment
0 0 0 Port	(Segment de) Port
0 0 1 Logical	(Segment) Logique
0 1 0 Network	(Segment de) Réseau
0 1 1 Symbolic	(Segment) Symbolique
1 0 0 Data / Extended symbolic	(Segment de)Données/ (Segment) Symbolique étendu
1 0 1 Reserved	Réservé
1 1 0 Reserved	Réservé
1 1 1 Reserved	Réservé

Figure 9 – Type de segment

4.1.9.3 Port segment

Le segment de port doit indiquer

- le port de communication par lequel quitter le nœud (exprimé sous la forme d'un nombre de 16 bits);
- l'adresse de liaison du prochain dispositif sur le chemin.

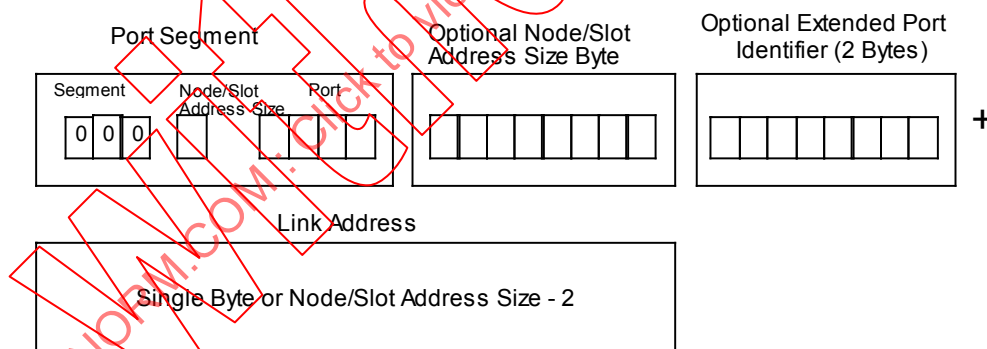
La Figure 10 montre la structure globale du segment de port.

Les bits 5 à 7 du premier octet doivent être zéro pour indiquer le type segment de port. Le bit 4, bit de taille d'adresse de nœud/intervalle de temps, doit être mis à 0 si l'adresse de liaison est d'un octet. Le bit 4 doit être mis à 1 si l'adresse de liaison est supérieure à un octet. Si l'adresse de liaison est supérieure à un octet, sa taille en octets doit être dans le deuxième octet du segment de port.

Les bits 0 à 3, l'identificateur de port, doivent indiquer le port par lequel quitter le nœud. L'identificateur de port doit spécifier un numéro de port ou un échappement à un identificateur de port étendu lorsque le module peut prendre en charge plus de 14 ports. Le numéro de port 0 doit être réservé. Le numéro de port 1 doit seulement être utilisé pour représenter un port de fond de panier. Si l'identificateur de port est 15, un champ de 16 bits appelé "identificateur de port étendu" doit être la partie suivante du segment de port; autrement, la valeur de l'identificateur de port doit être le numéro de port.

Le numéro de port doit être suivi de l'adresse de liaison dont le format dépend du type de port. Si l'adresse de liaison est supérieure à un octet, elle doit être bourrée afin que le segment de port entier soit un nombre entier d'octets. L'octet spécifiant la taille de l'adresse de liaison ne doit pas inclure l'octet de bourrage. L'octet de bourrage doit être mis à 0.

NOTE 1 Pour les types de port communs, l'adresse de liaison est un octet unique. Les autres types de port, tels que l'encapsulation TCP/IP, peuvent exiger une plus grande adresse de liaison (voir 4.3 et l'Article 11).



Légende

Anglais	Français
Port Segment	Segment de Port
Segment	Segment de Por
Node/Slot Address Size	Nœud/Position Taille adresse
Optional Node/Slot Address Size Byte	Ocet facultatifde taille adresse position/nœud
Optional Extended Port Identifier (2 Bytes)	Identificateur de Port étendu facultatif (2 octets)
Link Address	Adresse de liaison
Single Byte or Node/Slot Address Size – 2	Octet simple ou Taille adresse d nœud/posiiton -2

Figure 10 – Segment de port

Le format de l'Extended Port Identifier du Port Segment doit être utilisé seulement lorsqu'il y a plus de 14 ports possibles sur le dispositif. Le Port Segment doit toujours être condensé au format de Port Segment le plus petit possible. Des exemples de segments de port possibles sont montrés dans le Tableau 125.

Tableau 125 – Exemples de segments de port possibles

Segment de port	Notes
[02][06]	Port 2, ID de MAC 6
[15][0E][31][33][31][2E] [31][35][31][2E][31][33] [32][2E][35][35]	Adresse multioctet pour le port Ethernet 5, l'adresse de liaison 131.151.132.55 (adresse IP). L'adresse est définie comme une matrice de caractères.
[0F][12][00][01]	Port 18, ID de MAC 1

La partie adresse de liaison d'un segment de chemin de connexion TCP/IP doit être codée au sein du segment de port sous la forme d'une chaîne de caractères ASCII. La chaîne doit avoir l'une des formes suivantes:

- Adresse IP en notation avec point, par exemple "130.151.132.55" (voir IETF RFC 1117 pour le format des adresses IP);
- Adresse IP en notation avec point, suivie d'un séparateur ":", suivi du numéro de port TCP à utiliser à l'adresse IP spécifiée;
- Nom d'hôte, par exemple: "plc.type2.org". Le nom d'hôte doit être résolu par le biais d'une demande DNS à un serveur de nom (voir IETF RFC 1035 pour des informations relatives aux noms d'hôte et à la résolution des noms);
- Nom d'hôte, suivi d'un séparateur ":", suivi du numéro de port TCP à utiliser auprès de l'hôte spécifié.

Le numéro de port doit être représenté soit en hexadécimal, soit en décimal. L'hexadécimal doit être indiqué par un "0x" de tête. Lorsqu'un numéro de port est spécifié, il doit être utilisé plutôt que le numéro de port standard utilisé pour le protocole d'encapsulation (0xAF12). Seul le port 0xAF12 est garanti comme étant disponible dans un dispositif conforme à CP 2/2.

NOTE 2 D'autres numéros de port TCP peuvent être mis en œuvre; cependant, la présente spécification ne fournit pas de mécanisme pour déterminer quels numéros de port TCP sont pris en charge par un dispositif. L'utilisation d'autres numéros de port TCP est donc déconseillée. Le numéro de port TCP garanti, 0xAF12, a été réservé par l'Internet Assigned Numbers Authority (IANA) pour une utilisation par le protocole d'encapsulation.

Comme les segments de port doivent être alignés sur des mots, un octet de bourrage peut être requis à la fin de la chaîne. L'octet de bourrage doit être 0x00 et ne doit pas être compté dans le champ Optional Address Size du segment de port.

NOTE 3 Des exemples de segments de port pour TCP/IP sont montrés dans le Tableau 126 ci-dessous.

Tableau 126 – Exemples d'adresses de liaison TCP/IP

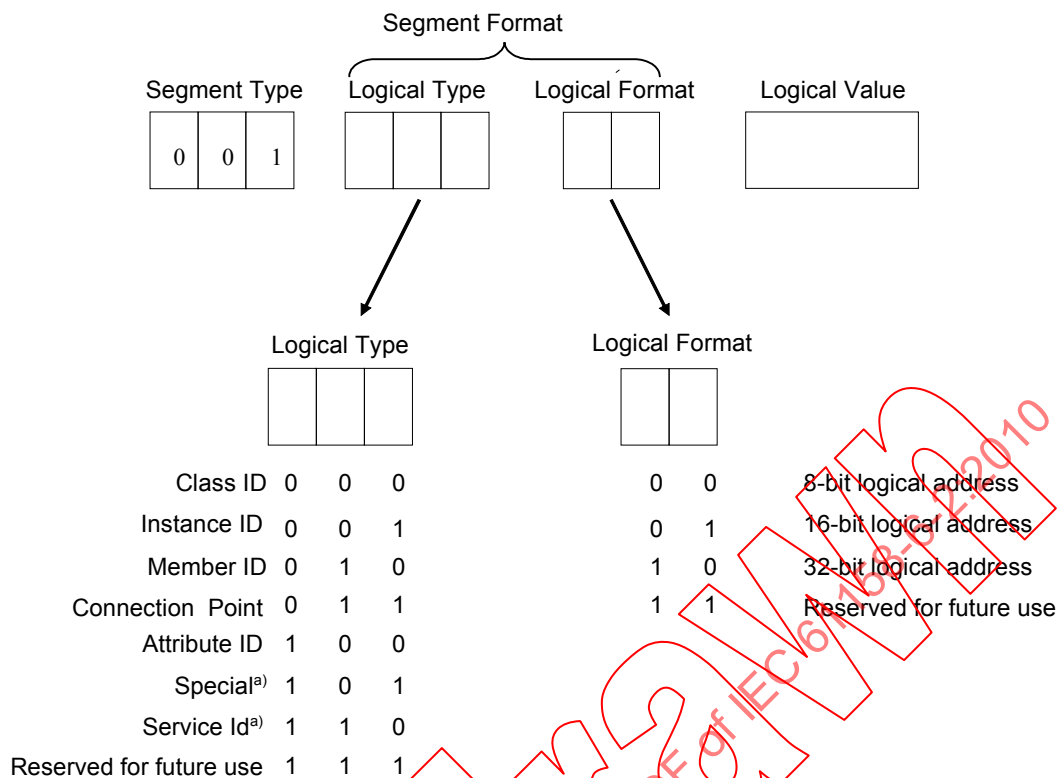
Segment de port	Adresse IP	Notes
[12][0D] [31][33][30][2E] [31][35][31][2E] [31][33][32][2E][31][00]	130.151.132.1	Adresse multioctet pour port 2, 13 octet string plus un octet de bourrage
[13][12] [70][6C][63][2E] [63][6F][6E][74][72][6F][6C][6E][65][74][2E] [6F][72][67]	plc. type2.org	Adresse multioctet pour port 3, 18 octet string, sans octet de bourrage
[16][15] [31][33][30][2E] [31][35][31][2E] [31][33][32][2E] [35][35][3A] [30][78][33][32][31][30][00]	130.151.132.55:0x3210	Adresse multioctet pour port 6, 21 octet string plus un octet de bourrage
[15][17] [70][6C][63][2E] [63][6F][6E][74][72][6F][6C][6E][65][74][2E] [6F][72][67][3A] [39][38][37][36][00]	plc. type2.org:9876	Adresse multioctet pour port 5, 32 octet string plus un octet de bourrage

4.1.9.4 Logical segment

4.1.9.4.1 Structure de Logical segment

Le segment logique sélectionne une entité adressable particulière au sein d'un dispositif (par exemple: classe d'objets, instance d'objet et attribut d'objet). Lorsque le segment logique est inclus au sein d'un Packed Path, la valeur logique (Logical Value) doit être accolée à l'octet de type de segment sans insérer de bourrage entre eux. Lorsque le segment logique est inclus au sein d'un Padded Path, les formats 16 bits et 32 bits doivent avoir un bourrage inséré entre l'octet de type de segment et la Logical Value (le format 8 bits est identique au Packed Path). L'octet de bourrage doit être mis à zéro.

La Figure 11 spécifie le codage d'un segment logique.

**Légende**

Anglais	Français
Segment Format	Format de segment
Segment Type	Type de segment
Logical Type	Type logique
Logical Format	Format logique
Logical Value	Valeur logique
Logical Type	Type logique
Logical Format	Format logique
Class ID	Identificateur de classe
Instance ID	Identificateur d'instance
Member ID	Identificateur de membre
Connection Point	Point de connexion
Attribute ID	Identificateur d'attribut
Special ^{a)}	Spécial
Service Id ^{a)}	Identificateur de service
Reserved for future use	Réservé pour un usage futur
8-bit logical address	Adresse logique 8 bits
16-bit logical address	Adresse logique 16 bits
32-bit logical address	Adresse logique 32 bits
Reserved for future use	Réservé pour un usage futur
^a Les types logiques <i>Special</i> et <i>Service ID</i> n'utilisent pas la définition d'adressage logique pour le format logique (Logical Format).	

^a Les types logiques *Special* et *Service ID* n'utilisent pas la définition d'adressage logique pour le format logique (Logical Format).

Figure 11 – Codage de segment logique

Les formats d'adresse logique 8 bits et 16 bits sont autorisés pour une utilisation avec tous les types logiques (Logical Type).

Le format d'adresse logique 32 bits est autorisé seulement pour les types logiques Instance ID et Connection Point. Il est interdit pour tout autre Logical Type (réservé pour usage futur).

Le Class ID (identificateur de classe) spécifie un type d'objet au sein d'un dispositif. L'Instance ID (identificateur d'instance) est un nombre entier attribué à un objet lorsqu'il est créé.. Instance 0, appelée l'instance de classe, spécifie la classe elle-même. Les attributs d'un objet sont une liste énumérée de valeurs visibles de l'extérieur. Attribute ID (identificateur d'attribut) spécifie quel attribut d'un objet est sélectionné. Les attributs complexes sont composés de membres. Member ID (identificateur de membre) spécifie le membre de l'attribut.

Le type logique Connection Point fournit des capacités d'adressage supplémentaires au-delà de l'adresse d'objet standard Class ID/Instance ID/Attribute ID/Member ID. Les classes d'objets doivent définir quand et comment ce composant d'adressage est utilisé (par exemple: une sous-instance d'un objet Assembly).

NOTE La CEI 61158-5-2:2010 décrit le modèle d'objet comportant la définition de classe, d'instance, d'attribut, de point de connexion et de membres d'une bibliothèque d'objets communs.

Le type logique Service ID a la définition suivante pour le format logique (Logical Format):

- 0 0 Segment "Service ID" de 8 bits (0x38)
- 0 1 Réservé pour usage futur (0x39)
- 1 0 Réservé pour usage futur (0x3A)
- 1 1 Réservé pour usage futur (0x3B)

Le type logique "Special" a la définition suivante pour le format logique (Logical Format):

- 0 0 Segment Electronic Key (clé électronique) (0x34)
- 0 1 Réservé pour usage futur (0x35)
- 1 0 Réservé pour usage futur (0x36)
- 1 1 Réservé pour usage futur (0x37)

4.1.9.4.2 Segment "Electronic Key"

Le segment clé électronique doit être utilisé pour vérifier le type de dispositif-cible ou dispositif routeur. La clé peut être incluse comme premier segment logique dans un chemin de connexion et doit être vérifiée par le Message Router du nœud de réception par rapport aux valeurs contenues dans l'instance 1 de son objet Identity avant que ne soient effectuées toutes vérifications d'adresse supplémentaires. Plusieurs segments de clé peuvent apparaître sur un message multi-saut envoyé vers une liaison distante en passant par des routeurs intermédiaires. Le format du segment de clé électronique doit être tel que montré dans le Tableau 127.

Tableau 127 – Format du segment "Electronic Key"

Paramètre	Format	Valeur
segment_type	USINT	toujours 0x34
electronic_key_type	USINT	toujours 0x04
vendor_ID	UINT	ils correspondent aux valeurs dans l'instance 1 de l'objet Identity.
product_type	UINT	
product_code	UINT	
major_revision: 7	USINT	
compatible_match: 1	USINT	
minor_revision	USINT	

Le paramètre `segment_type` pour un segment clé doit être 0x34. Le paramètre `electronic_key_type` doit déterminer le format du segment clé spécifique et doit être mis à 0x04. Toutes les autres valeurs de `electronic_key_type` doivent être réservées.

Le paramètre `vendor_ID` doit spécifier le vendeur de dispositif ou zéro si aucun vendeur spécifique n'est requis.

Le paramètre `product_type` doit spécifier une classe de produits tels qu'une entrée numérique ou des sorties analogiques. Le paramètre `product_type` doit être ignoré s'il est mis à zéro. Le paramètre `product_code` doit être spécifique au vendeur, chaque vendeur ayant un code unique. Le paramètre `product_code` doit être ignoré s'il est mis à zéro.

Les champs `major_revision` et `compatible_match` doivent être respectivement contenus dans les 7 bits de poids faible et le bit de poids fort du même octet.

Le paramètre `major_revision` doit spécifier le niveau de fonctionnalité d'un dispositif. Le paramètre `major_revision` doit être ignoré s'il est mis à zéro.

Le paramètre `compatible_match` doit modifier la fonction correspondante de la clé. Si le bit est effacé (= 0), tout paramètre non nul `vendor_ID`, `product_type`, `product_code`, `major_revision` et `minor_revision` doit correspondre exactement. Si le bit est mis (= 1), le dispositif peut accepter la clé pour tout dispositif qu'il peut émuler. Le niveau d'émulation est spécifique à chaque produit.

Le paramètre `minor_revision` doit spécifier la révision d'un dispositif qui n'altère pas le comportement visible du réseau. Une valeur de zéro doit spécifier que l'émetteur de la demande accepte toute révision mineure.

NOTE 2 La CEI 61158-5-2:2010 décrit l'objet Identity et les règles pour la mise à jour des champs de révision.

4.1.9.4.3 Exemples de segments logiques

Le Tableau 128 montre des exemples de segments logiques.

Tableau 128 – Exemples de segments logiques

Premier octet	Nom de segment logique	Exemples de formats bourrés (tels qu'émis)	Exemples de formats condensés (tels qu'émis)	Notes
0x20	8-bit class	[20][04]	[20][04]	classe 0x04 (objet Assembly)
0x21	16-bit class	[21][00][34][12]	[21][34][12]	classe 0x1234

Premier octet	Nom de segment logique	Exemples de formats bourrés (tels qu'émis)	Exemples de formats condensés (tels qu'émis)	Notes
0x24	8-bit instance	[24][04]	[24][04]	instance 0x04
0x25	16-bit instance	[25][00][34][12]	[25][34][12]	instance 0x1234
0x28	8-bit member	[28][04]	[28][04]	membre 0x04
0x29	16-bit member	[29][00][34][12]	[29][34][12]	membre 0x1234
0x2A	32-bit member			
0x2C	8-bit connection point	[2C][04]	[2C][04]	point de connexion 0x04
0x2D	16-bit connection point	[2D][00][34][12]	[2D][34][12]	point de connexion 0x1234
0x30	8-bit attribute	[30][04]	[30][04]	attribut 0x04
0x31	16-bit attribute	[31][00][34][12]	[31][34][12]	attribut 0x1234
0x34	electronic key	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	[34][04] [12][00] [EF][BE] [0E][0B] [83] [01]	ID de vendeur ID = 0x0012 type de produit = 0xBEEF code de produit = 0x0B0E révision majeure = 0x03 révision mineure = 0x01 accepter les clés compatibles = oui

4.1.9.5 Network segment

4.1.9.5.1 Structure de "Network segment"

Le type de segment (premier octet) d'un segment de réseau doit se situer dans la plage de 0x40 à 0x5F comme montré dans le Tableau 129 (les bits de poids fort doivent être 010). Le segment de réseau doit être utilisé pour spécifier les paramètres réseau qui peuvent être requis par un nœud pour émettre un message sur un réseau. Le segment de réseau doit précéder immédiatement le segment de port du dispositif auquel il s'applique. En d'autres termes, le segment de réseau doit être le premier élément dans le chemin que le dispositif reçoit.

Tableau 129 – Segments de réseau

Premier octet	Nom du segment de réseau
0x40	reserved (réservé)
0x41	schedule segment (segment programme)
0x42	fixed tag segment (segment à étiquette fixe)
0x43	production inhibit time (segment temps de neutralisation de production)
0x44 à 0x4F	reserved (réservé)
0x050	safety segment (segment sécurité)
0x051 à 0x05E	reserved (réservé)
0x5F	extended network segment (segment réseau étendu)

4.1.9.5.2 Schedule Segment

Le type de segment du segment de réseau de programme doit être 0x41. Le segment de réseau de programme doit spécifier

- un multiplicateur qui, lorsqu'il est multiplié par le temps de mise à jour réseau (NUT)⁵, donne le CM_API (intervalle réel entre paquets) du transport programmé;
- une phase qui détermine sur quelles valeurs du `Moderator.interval_count` émettre.

NOTE 1 La couche liaison de données définit le `Moderator.interval_count`, voir la CEI 61158-4-2:2010.

Ce segment doit être inclus dans le chemin vers un dispositif afin que chaque nœud intermédiaire puisse programmer le temps d'émission de liaison pour le trafic programmé ultérieur. Le multiplicateur doit avoir l'une des valeurs de 1, 2, 4, 8, 16, 32, 64 ou 128. La phase doit se situer dans la plage de 0 à (multiplicateur – 1). Le deuxième octet du segment de réseau de programme doit coder le multiplicateur et aussi la phase en les ajoutant ensemble.

NOTE 2 Si un transport produit tous les 64 NUT en commençant à `interval_count = 17`, la valeur codée est $17 + 64 = 81$.

Une valeur codée de zéro doit spécifier que le transport n'a pas encore reçu la permission d'utiliser la priorité programmée. Si un nœud reçoit une demande pour une connexion programmée dans laquelle n'existe aucun segment de réseau de programme ou le segment de réseau de programme existe, mais la valeur codée est 0, le nœud doit retourner le statut général = 0x01 et le statut étendu = 0x0317.

NOTE 3 Un émetteur de connexion est autorisé à utiliser la priorité programmée par le truchement de l'objet Scheduling (CEI 61158-5-2:2010).

4.1.9.5.3 Fixed Tag Segment

Le type de segment du segment de réseau à étiquette fixe doit être 0x42. Le segment de réseau à étiquette fixe doit spécifier l'étiquette fixe qui est à utiliser pour envoyer un message non connecté. Ce sous-type de segment doit précéder le segment de port dans le chemin. Si le segment à étiquette fixe est absent, l'étiquette fixe par défaut doit être utilisée.

NOTE 4 Le segment à étiquette fixe peut être utilisé à sein du chemin du service `Unconnected_Send` (voir 4.1.5.6) du Connection Manager pour envoyer des demandes à l'objet Keeper par le truchement du Management UCMM (voir la CEI 61158-5-2:2010).

4.1.9.5.4 Segment Production Inhibit Time

Le type de segment du segment temps de neutralisation de production doit être 0x43. Le deuxième octet doit spécifier le temps de neutralisation de production, qui est le temps minimum, en millisecondes, entre des émissions successives de données connectées pour la connexion spécifiée.

NOTE 5 Par exemple, si un temps de neutralisation de production de 10 millisecondes est spécifié, de nouvelles données doivent être envoyées au plus tôt 10 millisecondes après les données précédentes.

Le segment de temps de neutralisation de production doit être utilisé seulement pour les connexions de classe de transport 1 avec change-of-state sur les liaisons TCP/IP, pour imposer une limite au débit de paquets. Le segment de temps de neutralisation de production peut être envoyé à un dispositif final ou intermédiaire.

Lorsque le segment de temps de neutralisation de production est utilisé, le RPI doit déterminer quand renvoyer les données. Le RPI doit être supérieur au temps de neutralisation de production. Si le RPI est inférieur au temps de neutralisation de production, la réponse `Forward_Open` doit être retournée avec erreur (statut 0x01, statut étendu 0x11B). Si le segment de temps de neutralisation de production est omis, le producteur doit utiliser un temps de neutralisation de production par défaut égal à 1/4 du RPI. Si le temps de

⁵ NUT = Network Update Time

neutralisation de production est 0, il ne doit pas y avoir de limite sur la façon dont les données rapides peuvent être envoyées sur la connexion.

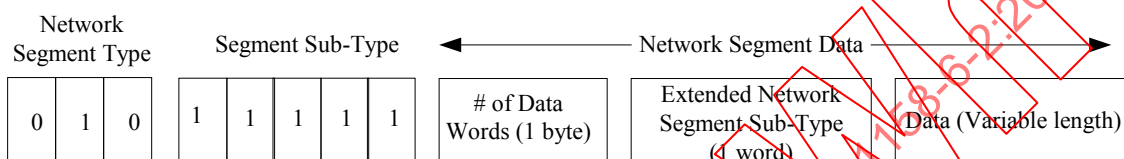
NOTE 6 La méthode d'émission des APDU sur Ethernet-TCP/IP est spécifiée en 4.3 et à l'Article 11.

4.1.9.5.5 Safety Segment (segment sécurité)

Pour le format, voir la CEI 61784-3-2.

4.1.9.5.6 Extended Network Segment (segment réseau étendu)

Le segment de réseau étendu permet la définition de sous-types supplémentaires de segment de réseau. Le premier mot des données est le sous-type de segment de réseau étendu. La structure du segment de réseau étendu est montrée à la Figure 12.



Légende

Anglais	Français
Network Segment Type	Type Segment de réseau
Segment Sub-Type	Sous-type de segment
Network Segment Data	Données de segment de réseau
# of Data Words (1 byte)	Nbr de mots de données (1 octet)
Extended Network Segment Sub-Type (1 word)	Sous-type de segment de réseau étendu (1 mot)
Data (Variable length)	Données (longueur variable)

Figure 12 – Segment de réseau étendu

Chaque sous-type étendu définit les données qui suivent. Les sous-types étendus sont énumérés dans le Tableau 130.

Tableau 130 – Définitions des sous-types étendus

Valeur de sous-type étendu	Définition de sous-type étendu
0 à 65 535	Réservé pour un usage futur

4.1.9.6 Symbolic segment

La plage de types de segment pour les segments symbolique doit être de 0x60 à 0x7F.

4.1.9.7 Data segment

4.1.9.7.1 Structure du segment de données

Le type de segment (premier octet) d'un segment de données doit se situer dans la plage de 0x80 à 0x8F (les bits de poids fort doivent être 1000). La présente norme définit seulement le

format du simple segment de données avec un type de segment de 0x80; tous les autres types de segment de données doivent être réservés (0x81 à 0x8F).

4.1.9.7.2 Simple segment de données

Le type de segment du simple segment de données doit être 0x80. Le deuxième octet doit représenter le nombre de mots de 16 bits des données de longueur variable. Les données de longueur variable doivent suivre le deuxième octet. Un chemin peut contenir plus d'un simple segment de données comme montré dans le Tableau 131.

Tableau 131 – Segment de données

Paramètre	Format	Valeur
segment_type	USINT	toujours 0x80
segment_size	USINT	taille de la matrice data[], en mots de 16 bits
data[]	UINT	

Les données peuvent être des informations de configuration pour un objet, des paramètres complémentaires nécessaires pour la connexion ou toute autre information. Le segment de données ne doit être interprété par aucun autre dispositif dans le chemin et, ainsi, seules les applications émettrices et cibles ont besoin de s'accorder sur son contenu.

4.1.9.8 Segment de symbole étendu

4.1.9.8.1 Structure du segment de symbole étendu

Le type de segment (premier octet) d'un segment de symbole étendu doit se situer dans la plage de 0x90 à 0x9F (les bits de poids fort doivent être 1001). La présente NORME définit seulement le format des segments de symbole étendu 0x91. Tous les autres types de segment de symbole étendu doivent être réservés (0x90 et 0x92 à 0x9F).

4.1.9.8.2 Structure du segment de symbole étendu ANSI

Le type de segment du segment de symbole étendu ANSI doit être 0x91. Le deuxième octet doit représenter le nombre caractères (de 8 bits) dans le symbole. Le symbole de longueur variable doit suivre le deuxième octet comme montré dans le Tableau 132.

Tableau 132 – Segment de l'ANSI_Extended_Symbol

Paramètre	Format	Valeur
segment_type	USINT	toujours 0x91
symbol_size	USINT	taille de la matrice symbol[]
symbol[]	USINT	
pad	USINT	seulement présent si symbol_size est impair, toujours mis à zéro

Le paramètre `symbol_size` doit être la taille de la matrice `symbol[]` en octets et ne doit pas être zéro. Le paramètre `symbol_size` ne doit pas compter l'octet `pad` (bourrage) s'il est inclus.

4.1.10 Codes de classe, d'attribut et de service

4.1.10.1 Plages des codes

4.1.10.1.1 Plages définies

Il doit y avoir trois catégories pour les plages d'adresse des Class ID, des Attribute ID et des Service Codes comme spécifié dans le Tableau 133.

Tableau 133 – Catégories d'adressage

Cette catégorie	Se réfère à
Open (Ouvverte)	Une valeur qui a la même signification pour tous les réalisateurs. Tous les services et classes d'objets définis dans la CEI 61158-5-2:2010 s'inscrivent dans cette catégorie.
Vendor-specific (Spécifique à un vendeur)	Une plage de valeurs spécifiques au vendeur d'un dispositif. Elles sont utilisées par les vendeurs pour étendre leurs dispositifs au-delà des options Open disponibles. Un vendeur gère en interne l'utilisation de valeurs dans cette plage. S'applique aux classes, attributs et services d'objets.
Object-class specific (Spécifique à une classe d'objets)	Un plage de valeurs dont la signification est définie par une classe d'objets. Cette plage s'applique aux définitions de Service Code.
NOTE Les valeurs ouvertes (OPEN) sont attribuées par ODVA, Inc.	

Les valeurs de Class ID, Attribute ID et Service Code doivent être telles que définies dans le Tableau 134 à Tableau 136.

4.1.10.1.2 Plage des ID de code de classe

Les valeurs de Class ID doivent être telles que montrées dans le Tableau 134. Le code de classe = 0x00 est réservé et ne doit pas être utilisé.

Tableau 134 – Plages des ID de code de classe

Plage (en hex)	Plage (en décimal)	Signification	Quantité
00	00	Reserved	1
01 - 63	01 - 99	Open	99
64 - C7	100 - 199	Vendor Specific	100
C8 - EF	200 - 239	Reserved	40
F0 - 2FF	240 à 767	Open	528
300 - 4FF	768 - 1 279	Vendor Specific	512
500 - FFFF	1 280 - 65 535	Reserved	64 256

4.1.10.1.3 Plages des ID d'attribut

Les valeurs de l'Attribute ID doivent être telles que montrées dans le Tableau 135. L'ID d'attribut = 0x00 est réservé et ne doit pas être utilisé.

Tableau 135 – Plages des ID d'attribut

Plage (en hex)	Plage (en décimal)	Signification	Quantité
00 à 63	00 à 99	Open	100
64 – C7	100 à 199	Vendor Specific	100
C8 – FF	200 à 255	Reserved	56
100 à 2FF	240 à 767	Open	512
300 - 4FF	768 à 1 279	Vendor Specific	512
500 à 8FF	1 280 à 2 303	Open	1 024
900 – CFF	2 304 à 3 327	Vendor Specific	1 024
D00 - FFFF	3 328 à 65 535	Réservé pour usage futur	62 208

4.1.10.1.4 Plages des codes de service

Le **Service code** (code de service) doit être une valeur hexadécimale unique attribuée à chaque service. Les valeurs de Service Code doivent être telles que montrées dans le Tableau 136. Le code de service = 0x00 est réservé et ne doit pas être utilisé. Les codes de services spécifiques à un objet doivent être uniques seulement au sein de la classe qui les définit.

Tableau 136 – Plages de codes de service

Plage (en hex)	Plage (en décimal)	Signification	Quantité
00	00	Réservés	1
01 - 31	01 - 49	Ouverts. Ils sont appelés <i>Common Services</i> (services communs). Ils sont définis dans les Articles concernant le Type 2 dans la CEI 61158-5-2:2010	49
32 - 4A	50 - 74	Spécifiques à un vendeur	25
4B - 63	75 - 99	Spécifiques à une classe d'objets	25
64 - 7F	100 - 127	Réservés pour un usage futur	28
80 - FF	128 - 255	Réservés pour des messages de réponse	128

4.1.10.2 Définitions des codes

4.1.10.2.1 Classes d'objets de communication

Le Tableau 137 définit les codes de classe pour les classes d'objets. Tous les autres codes de classe énumérés dans la plage "open" et non énumérés dans le Tableau 137 doivent être réservés. Les exigences pour ces objets sont définies dans la CEI 61158-5-2:2010.

Tableau 137 – Codes de classe

Code de classe (hex)	Code de classe (en décimal)	Nom d'objet
0x01	1	Objet "Identity"
0x02	2	Objet "Message Router"
0x03	3	Objet "DeviceNet" ^a
0x04	4	Objet "Assembly"
0x05	5	Objet "Connection"
0x06	6	Objet "Connection Manager"
0x0F	15	Objet "Parameter"
0x2B	43	Objet "Acknowledge Handler"
0x43	67	Objet "Time Sync"
0xF0	240	Objet "ControlNet" ^a
0xF1	241	Objet "Keeper" ^a
0xF2	242	Objet "Scheduling" ^a
0xF3	243	Objet "Connection Configuration" ^a
0xF5	245	Objet "TCP/IP Interface" ^a
0xF6	246	Objet "Ethernet Link" ^a
^a Cette classe d'objets est partie intégrante de la gestion de système.		

4.1.10.2.2 Attributs de classe prédéfinis

Sept ID d'attribut de classe prédéfinis doivent être réservés et ces attributs de classe prédéfinis/réservés doivent avoir leurs définitions énumérées dans le Tableau 138. Comme ces attributs sont réservés, les numéros d'ID d'attribut de classe 1 à 7 doivent toujours être réservés. Par conséquent, si un attribut de classe est ajouté à une spécification de classe d'objets, il doit commencer par l'ID d'attribut n° 8.

Tableau 138 – Attributs de classe réservés pour toutes les définitions de classes d'objets

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
1	Revision	UINT	La valeur de départ attribuée à cet attribut est un (1). Si des mises à jour nécessitant une augmentation de cette valeur sont effectuées, la valeur de cet attribut augmente de un (1).
2	Max Instance	UINT ou UDINT	La définition de classe doit définir le type de données utilisé.
3	Number of Instances	UINT ou UDINT	La définition de classe doit définir le type de données utilisé.
4	Optional Attribute List	STRUCT of	
	Number of Attributes	UINT	
	Optional attributes	ARRAY of UINT	
5	Optional service list	STRUCT of	
	Number services	UINT	

ID d'attribut	Nom	Type de donnée	Sémantique des valeurs
	Optional services	ARRAY of UINT	
6	Maximum ID Number Class Attributes	UINT	
7	Maximum ID Number Instance Attributes	UINT	

4.1.10.2.3 Codes d'OM_Service

Les codes des services communs pour les objets de réseau de contrôle déterministe doivent être tels que montrés dans le Tableau 139.

NOTE Le Tableau 139 énumère les codes et noms des services communs alors que la CEI 61158-5-2:2010 donne une description générale de chaque service.

Tableau 139 – Liste des services communs

Code de service commun	Nom de service commun
0x00	réservé
0x01	Get_Attribute_All
0x02	Set_Attribute_All
0x03	Get_Attribute_List
0x04	Set_Attribute_List
0x05	Reset
0x06	Start
0x07	Stop
0x08	Create
0x09	Delete
0x0A à 0x0C	réservé
0x0D	Apply_Attributes
0x0E	Get_Attribute_Single
0x0F	réservé
0x10	Set_Attribute_Single
0x11	Find_Next_Object_Instance
0x12 à 0x14	réservé
0x15	Restore
0x16	Save
0x17	NOP (No operation, aucune opération)
0x18 à 0x1B	réservé
0x1C	Group_Sync
0x1D à 0x31	réservé

Les codes des services spécifiques à un objet pour l'objet Acknowledge Handler doivent être tels que montrés dans le Tableau 140.

Tableau 140 – Liste des services spécifiques à un objet Acknowledge Handler

Code de service spécifique à un objet	Nom du service
0x4B	Add_AckData_Path
0x4C	Remove_AckData_Path

Le code du service spécifique à un objet pour l'objet Parameter doit être tel que montré dans le Tableau 141.

Tableau 141 – Liste des services spécifiques à un objet Parameter

Code de service spécifique à un objet	Nom du service
0x4B	Get_Enum_String

4.1.10.2.4 Codes de CM_Service

Les codes des services spécifiques au Connection Manager doivent être tels que montrés dans le Tableau 142.

NOTE Le Tableau 142 énumère les codes et noms des services alors que la CEI 61158-5-2:2010 donne une description générale de chaque service.

Tableau 142 – Services spécifiques au Connection Manager

Code de service	Nom du service
0x4E	Forward_Close
0x52	Unconnected_Send
0x54	Forward_Open
0x56	Get_Connection_Data
0x57	Search_Connection_Data
0x5A	Get_Object_Owner
0x5B	Large_Forward_Open

4.1.10.2.5 Codes de CO_Service

Les codes des services spécifiques à l'objet Connection doivent être tels que montrés dans le Tableau 143.

NOTE Le Tableau 143 énumère les codes et noms des services alors que la CEI 61158-5-2:2010 donne une description générale de chaque service.

Tableau 143 – Services spécifiques à l'objet Connection

Code de service	Nom du service
0x4B	Connection_Bind
0x4C	Producing_Application_Lookup
0x4E	SafetyClose
0x54	SafetyOpen

4.1.10.3 Types de dispositif

Afin de permettre l'interopérabilité et l'intervention, un type de dispositif doit être utilisé pour identifier des dispositifs semblables qui

- montrent le même comportement;
- produisent et/ou consomment le même jeu de données;
- contiennent le même jeu d'attributs configurables.

La définition formelle de ces informations est connue sous le nom de Device Profile (profil de dispositif): tous les dispositifs ayant le même numéro de type de dispositif doivent satisfaire aux exigences spécifiées dans le Device Profile pour le type de dispositif en question.

Le "Device Type" (type de dispositif) est un attribut d'instance requis de l'objet Identity tel que décrit dans la CEI 61158-5-2:2010.

Les types de dispositif doivent être soit définis publiquement, soit spécifiques au vendeur. Le Tableau 144 révèle le plan de numérotation à utiliser pour numéroté les types de dispositif.

Tableau 144 – Numérotation du type de dispositif

Plage (en hex)	Type	Quantité
00 - 63	Défini publiquement – Réserve	100
64 - C7	Spécifique à un vendeur	100
C8 - FF	Réserve	56
100 - 2FF	Défini publiquement – Réserve	512
300 - 4FF	Spécifique à un vendeur	512
500 - FFFF	Réserve	64 256

Tous les types de dispositif définis publiquement doivent avoir la même signification pour tous les réalisateurs et sont réservés. Le type Generic Device (dispositif générique) (numéro de type = 0x00) doit définir un dispositif qui ne correspond à aucun des types de dispositif définis.

NOTE La définition réelle des types de dispositif définis publiquement et des Device Profile correspondants ne relève pas du domaine d'application de la présente norme. Ils sont définis par ODVA, Inc.

Les types de dispositif spécifiques à un vendeur peuvent être développés et les vendeurs peuvent ne pas les publier. Chaque vendeur doit maintenir ses propres types de dispositif spécifiques à un vendeur et sa propre allocation de numéros correspondante.

4.1.11 Codes d'erreur

4.1.11.1 Erreurs de CM

Les codes d'erreur sont retournés avec la réponse à une demande de service Connection Manager qui a abouti à une erreur. Les codes d'erreur doivent être utilisés pour aider à diagnostiquer le problème avec une demande de service. Le code d'erreur doit être divisé en un statut général de 8 bits et un ou plusieurs mots de 16 bits de statut étendu. Sauf spécification contraire, seul le premier mot de statut étendu doit être requis. Des mots supplémentaires de statut étendu peuvent être utilisés pour spécifier des informations supplémentaires de débogage spécifiques à un module. Tous les dispositifs qui produisent des messages doivent être capables de gérer plusieurs mots de statut étendu.

Le Tableau 145 donne un résumé des codes d'erreur disponibles.

Tableau 145 – Codes d'erreur d'une demande de service de Connection Manager

Statut général	Statut étendu	Explication
0x00		Service achevé avec succès
0x01	0x0000 à 0x00FF	Désuet
0x01	0x0100	CONNECTION IN USE OR DUPLICATE FORWARD OPEN (connexion en utilisation ou doublon de forward_open) Ce code de statut étendu doit être retourné lorsqu'un émetteur tente d'établir une connexion vers une cible avec laquelle l'émetteur peut avoir déjà établi une connexion (doublon de Forward_Open – voir 4.1.5.3).
0x01	0x0101 à 0x0102	Réservé
0x01	0x0103	TRANSPORT CLASS AND TRIGGER COMBINATION NOT SUPPORTED (combinaison de classe de transport et de déclencheur non prise en charge) Une combinaison de classe de transport et de déclencheur a été spécifiée alors qu'elle n'est pas prise en charge par la cible. Les routeurs doivent faire échouer la connexion basée sur la combinaison de classe de transport et de déclencheur. Seules les cibles doivent retourner ce code de statut étendu.
0x01	0x0104 à 0x0105	Réservé
0x01	0x0106	OWNERSHIP CONFLICT (conflit de propriété) La connexion ne peut pas être établie, car une autre connexion a alloué en exclusivité certaines des ressources requises pour cette connexion. Un exemple en serait que seule une connexion de propriétaire exclusif peut contrôler un point de sortie sur un module E/S. Si une seconde connexion de propriétaire exclusif (ou une connexion de propriétaire redondant) est tentée, cette erreur doit être retournée. Ce code de statut étendu doit être retourné seulement par un nœud-cible.
0x01	0x0107	TARGET CONNECTION NOT FOUND (connexion-cible introuvable) Ce code de statut étendu doit être retourné en réponse à la demande forward_close lorsque la connexion à fermer est introuvable au nœud-cible. Ce code de statut étendu doit être retourné seulement par un nœud-cible. Les routeurs ne doivent pas générer ce code de statut étendu. Si la connexion spécifiée est introuvable au nœud intermédiaire, la demande de fermeture doit encore être transmise par le chemin spécifié dans la demande Forward_Close.

Statut général	Statut étendu	Explication
0x01	0x0108	INVALID NETWORK CONNECTION PARAMETER (paramètre de connexion réseau non valide) Ce code d'erreur étendu doit être retourné comme résultat de la spécification d'un type de connexion, d'une priorité de connexion, d'un propriétaire redondant ou d'une connexion fixe/variable que l'application-cible ne prend pas en charge. Seul un nœud-cible doit retourner ce code de statut étendu.
0x01	0x0109	INVALID CONNECTION SIZE (taille de connexion non valide) Ce code de statut étendu est retourné lorsque la cible ou le routeur ne prend pas en charge la taille de connexion spécifiée. Cela peut se produire au niveau d'une cible parce que la taille ne correspond pas à la taille requise pour une connexion de taille fixe. Cela pourrait se produire au niveau d'un routeur si la taille demandée est trop grande pour le réseau spécifié. Un statut supplémentaire peut suivre en indiquant la taille de connexion maximale prise en charge par le nœud répondeur. Le mot de statut supplémentaire est requis lorsqu'il est émis en réponse à Large_Forward_Open.
0x01	0x010A à 0x010F	Réservé
0x01	0x0110	TARGET FOR CONNECTION NOT CONFIGURED (cible pour la connexion non configurée) Ce code de statut étendu doit être retourné lorsqu'une connexion est demandée à une application-cible qui n'a pas été configurée et la demande de connexion ne contient pas un segment de données pour la configuration (voir 4.1.9.7). Seul un nœud-cible doit retourner ce code de statut étendu.
0x01	0x0111	RPI NOT SUPPORTED (RPI non pris en charge) Ce code de statut étendu doit être retourné si le dispositif ne peut pas prendre en charge le RPI demandé dans le sens O⇒T ou dans le sens T⇒O. Ce code de statut étendu doit également être utilisé si le multiplicateur de temporisation de connexion (connection time-out multiplier) produit une valeur de temporisation qui n'est pas prise en charge par le dispositif ou bien le temps de neutralisation de production n'est pas valide.
0x01	0x0112	Reserved
0x01	0x0113	OUT OF CONNECTIONS (à court de connexions) Le Connection Manager ne peut pas prendre d'autres connexions. Le nombre maximum des connexions prises en charge par le Connection Manager a déjà été créé.
0x01	0x0114	VENDOR ID OR PRODUCT CODE MISMATCH (discordance d'ID de vendeur ou de code de produit) Le code de produit ou l'ID de vendeur spécifié dans le segment logique "clé électronique" ne concorde pas le code de produit ou l'ID de vendeur du dispositif-cible.
0x01	0x0115	PRODUCT TYPE MISMATCH (discordance de type de produit) Le Product Type spécifié dans le segment logique "clé électronique" ne concorde pas avec le Product Type du dispositif-cible.
0x01	0x0116	REVISION MISMATCH (discordance de révision) Les révisions majeure et mineure spécifiées dans le segment logique "clé électronique" ne correspondent pas à une révision valide du dispositif-cible.

Statut général	Statut étendu	Explication
0x01	0x0117	INVALID PRODUCED OR CONSUMED APPLICATION PATH (chemin d'application produit ou consommé non valide) Le chemin d'application produit ou consommé spécifié dans le chemin de connexion ne correspond pas à un chemin d'application produit ou consommé valide au sein de l'application-cible. Cette erreur pourrait également être retournée si un chemin d'application produit ou consommé était requis, mais n'était pas fourni par une demande de connexion.
0x01	0x0118	INVALID OR INCONSISTENT CONFIGURATION APPLICATION PATH (chemin d'application de configuration non valide ou incohérent) Un chemin d'application spécifié pour les données de configuration ne correspond pas à une application de configuration ou n'est pas cohérent avec les chemins d'application consommés ou produits. Par exemple, le chemin de connexion spécifie des données de configuration de type float (flottantes) alors que les chemins produits ou consommés spécifient des données de type Integer (entières).
0x01	0x0119	NON-LISTEN ONLY CONNECTION NOT OPENED (connexion qui ne soit pas en écoute seule non ouverte) Le demande de connexion échoue parce qu'il n'y a pas de types actuellement ouverts de connexion qui ne soit pas en écoute seule. Voir la CEI 61158-5-2:2010, 6.3.1.4.5, pour une description des types de connexion d'application. Le code de statut étendu doit être retourné lorsqu'une tentative est faite d'établir un type de connexion en écoute seule vers une cible qui n'a aucune connexion qui ne soit pas en écoute seule déjà établie.
0x01	0x011A	TARGET OBJECT OUT OF CONNECTIONS (objet-cible à court de connexions) Le nombre maximum de connexions prises en charge par cette instance de l'objet-cible a été dépassé. Par exemple, le Connection Manager pourrait prendre en charge 20 connexions alors que l'objet-cible peut seulement prendre en charge 10 connexions. À la 11ème demande de connexion vers l'objet-cible, ce code de statut étendu serait utilisé pour signifier que le nombre maximum de connexions existe déjà en l'objet-cible.
0x01	0x011B	RPI IS SMALLER THAN THE PRODUCTION INHIBIT TIME (le RPI est inférieur au temps de neutralisation de production) Le RPI dans le sens Cible vers Émetteur est plus petit que le temps de neutralisation de production dans le sens Cible vers Émetteur.
0x01	0x011C à 0x0202	Reserved
0x01	0x0203	CONNECTION TIMED OUT (connexion expirée) Ce code de statut étendu doit se produire lorsqu'un client tente d'envoyer un message connecté sur une connexion qui a été temporisée. Ce code de statut étendu doit seulement se produire localement au niveau du nœud producteur.
0x01	0x0204	UNCONNECTED REQUEST TIMED OUT L'erreur Unconnected Request Timed Out (demande non connectée expirée) doit se produire lorsque l'UCMM temporise avant qu'une réponse ne soit reçue. Cela peut se produire pour un service Unconnected_Send, Forward_Open ou Forward_Close. Cela signifie typiquement que l'UCMM a tenté un nombre de fois spécifique à une liaison en utilisant un temporisateur de répétitions de tentative spécifique à une liaison et n'a pas reçu un acquittement ou une réponse. Cela peut résulter d'un engorgement au nœud de destination ou peut résulter du fait qu'un nœud ne soit pas alimenté en énergie ou présent. Ce code de statut étendu doit être retourné par le nœud émetteur ou n'importe quel nœud intermédiaire.

Statut général	Statut étendu	Explication
0x01	0x0205	PARAMETER ERROR IN UNCONNECTED REQUEST SERVICE (erreur de paramètre dans le service de demande non connecté) Par exemple, cela doit être causé par une combinaison de Connection Tick Time (voir CEI 61158-5-2:2010, 6.2.3.2.1.7) et de Connection time-out dans un service Unconnected_Send, Forward_Open ou Forward_Close qui n'est pas prise en charge par un nœud intermédiaire.
0x01	0x0206	MESSAGE TOO LARGE FOR UNCONNECTED_SEND SERVICE (message trop gros pour le service Unconnected_Send) Cela doit être apparaître lorsque le service Unconnected_Send est trop gros pour être envoyé sur un réseau.
0x01	0x0207	UNCONNECTED ACKNOWLEDGE WITHOUT RESPONSE (acquiescement non connecté sans réponse) Le message a été envoyé par le service de messages non connecté et un acquiescement a été reçu, mais le message de réponse de données n'a pas été reçu.
0x01	0x0208 à 0x0300	Réservé
0x01	0x0301	NO BUFFER MEMORY AVAILABLE (pas de mémoire-tampon disponible) Le code de statut étendu doit se produire lorsque la mémoire-tampon des connexions disponible est insuffisante dans les dispositifs-cibles ou tout dispositif-routeur. Les routeurs et les nœuds-cibles doivent retourner cette erreur.
0x01	0x0302	LINK TRANSMIT TIME NOT AVAILABLE FOR DATA (temps d'émission de liaison non disponible pour les données) Ce code de statut étendu doit être retourné par tout dispositif du chemin qui est un producteur et ne peut pas allouer un temps d'émission de liaison suffisant pour la connexion sur sa liaison. Cela peut se produire en n'importe quel nœud. Cela peut se produire seulement pour les connexions qui sont spécifiées comme étant de priorité programmée.
0x01	0x0303	NO CONSUMED CONNECTION ID FILTER AVAILABLE (aucun filtre d'ID de connexion consommée disponible) Tout dispositif du chemin qui contient un consommateur de liaisons pour la connexion et n'a pas un filtre disponible de consumed_connection_id disponible doit retourner ce code de statut étendu.
0x01	0x0304	NOT CONFIGURED TO SEND SCHEDULED PRIORITY DATA (non configuré pour envoyer des données de priorité programmée) S'il lui est demandé d'établir une connexion qui spécifie une priorité programmée, tout dispositif qui est incapable d'envoyer des données dans la partie programmée de l'intervalle de temps de mise à jour de réseau doit retourner ce code de statut étendu. Par exemple, sur CP 2/1, ce code doit être retourné par un nœud dont l'ID de MAC est supérieur au nœud programmé maximal (SMAX).
0x01	0x0305	SCHEDULE SIGNATURE MISMATCH (discordance de signature de programme) Ce code de statut étendu doit être retourné lorsque les informations de programmation de connexion dans le dispositif émetteur ne sont pas cohérentes avec les informations de programmation de connexion sur le réseau-cible.
0x01	0x0306	SCHEDULE SIGNATURE VALIDATION NOT POSSIBLE (validation impossible de la signature de programme) Ce code de statut étendu doit être retourné lorsque les informations de programmation de connexion dans le dispositif émetteur ne peuvent pas être validées sur le réseau-cible. Par exemple, sur CP 2/1, ce code doit être retourné lorsqu'il n'y a pas de Keeper dans l'état maître.
0x01	0x0307 à 0x0310	Réservé

Statut général	Statut étendu	Explication
0x01	0x0311	PORT NOT AVAILABLE (port non disponible) Un port spécifié dans un segment de port n'est pas disponible ou n'existe pas.
0x01	0x0312	LINK ADDRESS NOT VALID (adresse de liaison non valide) Adresse de liaison spécifiée dans un segment de port non valide Ce code de statut étendu est le résultat d'un segment de port qui spécifie une adresse de liaison qui n'est pas valide pour le type de réseau-cible. Le code de statut étendu ne doit pas être utilisé pour des adresses de liaison qui sont valides pour le type de réseau-cible mais ne répondent pas.
0x01	0x0313 à 0x0314	Réservé
0x01	0x0315	INVALID SEGMENT IN CONNECTION PATH (segment non valide dans le chemin de connexion) Type de segment ou valeur de segment non valide dans le chemin de connexion Ce code de statut étendu résulte de l'incapacité d'un dispositif à décoder le chemin de connexion. Cela pourrait être dû à un type de chemin non reconnu, à un type de segment apparaissant intempestivement ou à une myriade d'autres problèmes apparaissant dans le chemin de connexion.
0x01	0x0316	ERROR IN FORWARD CLOSE SERVICE CONNECTION PATH MISMATCH (erreur de discordance du chemin de connexion du service Forward_Close) Le chemin de connexion dans le service Forward_Close ne concorde pas avec le chemin de connexion dans la connexion fermée.
0x01	0x0317	SCHEDULING NOT SPECIFIED (programmation non spécifiée) Le segment de réseau Schedule n'était pas présent ou bien la valeur codée dans le segment de réseau Schedule n'est pas valide (0).
0x01	0x0318	LINK ADDRESS TO SELF INVALID (adresse de liaison en boucle non valide) Dans certaines conditions (en fonction du dispositif), une adresse de liaison dans le segment de port qui pointe vers le même dispositif (bouclage vers soi) est non valide.
0x01	0x0319	SECONDARY RESOURCES UNAVAILABLE (ressources secondaires non disponibles) Dans un système redondant double châssis, une demande de connexion qui est faite au système primaire doit être dupliquée sur le système secondaire. Si le système secondaire est incapable de dupliquer la demande de connexion, ce code de statut étendu doit être retourné.
0x01	0x031A	RACK CONNECTION ALREADY ESTABLISHED (connexion de baie déjà établie) Une demande pour une connexion de module a été refusée parce qu'une partie des données correspondantes est déjà incluse dans une connexion de baie.
0x01	0x031B	MODULE CONNECTION ALREADY ESTABLISHED (connexion de module déjà établie) Une demande pour une connexion de baie a été refusée parce qu'une partie des données correspondantes est déjà incluse dans une connexion de module.
0x01	0x031C	MISCELLANEOUS (divers) Ce statut étendu est retourné lorsqu'aucun autre code de statut étendu ne s'applique pour une erreur liée à la connexion.

Statut général	Statut étendu	Explication
0x01	0x031D	REDUNDANT CONNECTION MISMATCH (discordance de connexion redondante) Ce code de statut étendu doit être retourné quand les champs ci-après ne concordent pas lors d'une tentative d'établissement d'une connexion de propriétaire redondant au même chemin-cible: O->T_RPI; O->T_connection_parameters; T->O_RPI; T->O_connection_parameters; xport_type_and_trigger.
0x01	0x031E	NO MORE USER CONFIGURABLE LINK CONSUMER RESOURCES AVAILABLE IN THE PRODUCING MODULE (plus aucune autre ressource de consommateur de liaison configurable par l'utilisateur disponible dans le module producteur) Une cible doit retourner ce statut étendu lorsque le nombre configuré de consommateurs pour une application productrice est déjà en cours d'utilisation.
0x01	0x031F	NO MORE USER CONFIGURABLE LINK CONSUMER RESOURCES AVAILABLE IN THE PRODUCING MODULE (plus aucune autre ressource de consommateur de liaison configurable par l'utilisateur disponible dans le module producteur) Une cible doit retourner ce statut étendu lorsqu'il n'y a pas de consommateurs configurés à utiliser par une application productrice.
0x01	0x0320 à 0x07FF	Spécifique à un vendeur
0x01	0x0800	La liaison réseau dans le chemin vers le module est hors ligne.
0x01	0x0801 à 0x080F	Réservé
0x01	0x0810	NO TARGET APPLICATION DATA AVAILABLE (aucune donnée d'application-cible disponible) Ce code de statut étendu est retourné lorsque l'application-cible n'a pas de données valides à produire pour la connexion demandée. Seul le côté cible d'une connexion doit retourner ce code de statut étendu.
0x01	0x0811	NO ORIGINATOR APPLICATION DATA AVAILABLE (aucune donnée d'application d'émetteur disponible) Ce code de statut étendu est retourné lorsque l'application d'émetteur n'a pas de données valides à produire pour la connexion demandée. Seul le côté émetteur d'une connexion doit indiquer ce code de statut étendu.
0x01	0x0812	NODE ADDRESS HAS CHANGED SINCE THE NETWORK WAS SCHEDULED. (l'adresse de nœud a changé depuis que le réseau a été programmé.) Un routeur sur un réseau programmé (CP 2/1 par exemple) a une adresse de nœud différente de la valeur configurée dans l'émetteur de la connexion.
0x01	0x0813	NOT CONFIGURED FOR OFF-SUBNET MULTICAST (pas configuré pour une multidiffusion hors sous-réseau) Une connexion de multidiffusion a été demandée entre un producteur et un consommateur qui se situent sur des sous-réseaux différents et le producteur n'est pas configuré pour une multidiffusion hors sous-réseau.
0x01	0x0814	INVALID PRODUCE/CONSUME DATA FORMAT (format non valide de données de production/consommation) Les informations dans le segment de données indiquent que le format des données produites et/ou consommées n'est pas valide.

Statut général	Statut étendu	Explication
0x01	0x0815 à 0xFCFF	Reserved
0x09	Index to Element (index vers élément)	ERROR IN DATA SEGMENT (erreur dans le segment de données). Ce code de statut général doit être retourné lorsqu'il y a une erreur dans le segment de données dans un forward open. Le statut étendu doit être l'index vers l'endroit où l'erreur a été rencontrée dans le segment de donnée (voir 4.1.9.7).
0x0C	Optional (Facultatif)	OBJECT STATE ERROR (erreur d'état d'objet) Ce code de statut général doit être retourné lorsque l'état de l'objet-cible de la connexion empêche que la demande de service soit gérée. Le statut étendu rend compte de l'état actuel de l'objet. Le statut étendu est facultatif. Par exemple, un objet (application) cible de la connexion peut avoir besoin d'être dans un mode d'édition avant que des attributs ne puissent être fixés. Cela est différent du rejet d'un service du fait de l'état du dispositif.
0x10	Optional (Facultatif)	DEVICE STATE ERROR (erreur d'état de dispositif) Ce code de statut général doit être retourné lorsque l'état du dispositif empêche que la demande de service soit gérée. Le statut étendu rend compte de l'état actuel du dispositif. Le statut étendu est facultatif. Par exemple, un dispositif de commande peut avoir un commutateur à clé qui, lorsqu'il est mis dans l'état "hard run" (marche dure), fait échouer des demandes de service à plusieurs objets (à savoir éditions de programme). Ce code de statut général serait alors retourné.
0x13	None (Aucun)	Pas assez de données. Le service n'a pas fourni assez de données pour accomplir l'opération spécifiée.
0x15	None (Aucun)	Trop de données. Le service a fourni plus de données qu'il n'en était prévu.
NOTE 1 Le mot "n/a" dans la colonne "Statut étendu" signifie qu'aucun autre statut étendu n'est tenu d'être retourné pour le code de statut général particulier. NOTE 2 Le mot "facultatif" dans la colonne "Statut étendu" signifie que si les informations de Statut étendu sont utilisées, le premier mot de ce statut étendu est déjà défini et un statut étendu défini par l'utilisateur doit commencer par le second mot de statut étendu.		

4.1.11.2 Erreurs d'OM

4.1.11.2.1 Format du statut général

Le statut général utilisé dans les primitives de service est spécifié dans le Tableau 146.

Tableau 146 – Codes de statut général

Status code	Nom	Description et signification du code de statut
0x00	Success (succès)	Le service a été accompli avec succès par l'objet spécifié.
0x01	Connection failure (échec de connexion)	Un service lié à la connexion a failli le long du chemin de connexion.
0x02	Resource unavailable (ressource non disponible)	Les ressources nécessaires pour que l'objet exécute le service demandé n'étaient pas disponibles.
0x03	Invalid parameter value (valeur de paramètre invalide)	Voir le code de statut 0x20, qui est la valeur préférentielle à utiliser pour cet état.
0x04	Path segment error (erreur de segment de chemin)	L'identificateur de segment de chemin ou la syntaxe de segment n'était pas compris(e) par le nœud de traitement. Le traitement de chemin doit cesser en cas d'erreur de segment de chemin.

Status code	Nom	Description et signification du code de statut
0x05	Path destination unknown (destination de chemin inconnue)	Le chemin fait référence à une classe, une instance, un élément de structure d'objet qui n'est pas connu(e) ou n'est pas contenu(e) dans le nœud de traitement. Le traitement de chemin doit cesser en cas d'erreur de destination inconnue.
0x06	Partial transfer (transfert partiel)	Une partie seulement des données prévues a été transférée.
0x07	Connection lost (connexion perdue)	La connexion de messagerie a été perdue.
0x08	Service not supported (service non pris en charge)	Le service demandé n'était pas mis en œuvre ou n'était pas défini pour cette instance de classe ou d'objet.
0x09	Invalid attribute value (valeur d'attribut non valide)	Données d'attribut invalides détectées
0x0A	Attribute list error (erreur de liste d'attributs)	Un attribut dans la réponse Get_Attribute_List ou Set_Attribute_List a un statut non nul.
0x0B	Already in requested mode/state (déjà dans le mode/état demandé)	L'objet est déjà dans le mode/état demandé par le service.
0x0C	Object state conflict (conflit d'état d'objet)	L'objet ne peut pas accomplir le service demandé dans son mode/état actuel.
0x0D	Object already exists (L'objet existe déjà)	L'instance demandée de l'objet à créer existe déjà.
0x0E	Attribute not settable (attribut non réglable)	Une demande de modifier un attribut non modifiable a été reçue.
0x0F	Privilege violation (violation de privilège)	Une vérification de permission/privilège a échoué.
0x10	Device state conflict (conflit d'état de dispositif)	Le mode/état courant du dispositif interdit l'exécution du service demandé.
0x11	Response data too large (données de réponse trop volumineuses)	Les données à émettre dans le tampon de réponses sont plus volumineuses que le tampon de réponses alloué.
0x12	Fragmentation of a primitive value (fragmentation d'une valeur de primitive)	Le service spécifiait une opération qui va fragmenter une valeur de données de primitive, c'est-à-dire, une moitié de type de données REAL.
0x13	Not enough data (Pas assez de données)	Le service n'a pas fourni assez de données pour accomplir l'opération spécifiée.
0x14	Attribute not supported (Attribut non pris en charge)	L'attribut spécifié dans la demande n'est pas pris en charge.
0x15	Too much data (Trop de données)	Le service a fourni plus de données qu'il n'en était prévu.
0x16	Object does not exist (L'objet n'existe pas)	L'objet spécifié n'existe pas dans le dispositif.
0x17	Service fragmentation sequence not in progress (séquence de fragmentation de service pas en cours)	La séquence de fragmentation pour ce service n'est pas active actuellement pour ces données.
0x18	No stored attribute data (Aucune donnée d'attribut stockée)	Les données d'attribut de cet objet n'étaient pas sauvegardées préalablement au service demandé.
0x19	Store operation failure (Échec de l'opération de stockage)	Les données d'attribut de cet objet n'étaient pas sauvegardées en raison d'une défaillance au cours de la tentative.
0x1A	Routing failure, request packet too large for network (Échec d'acheminement, paquet de demande trop volumineux pour le réseau)	Le paquet de demande de service était trop volumineux pour l'émission sur un réseau dans le chemin vers la destination. Le dispositif routeur était forcé d'abandonner le service.

Status code	Nom	Description et signification du code de statut
0x1B	Routing failure, response packet too large for network (Échec d'acheminement, paquet de réponse trop volumineux pour le réseau)	Le paquet de réponse de service était trop volumineux pour l'émission sur un réseau dans le chemin en provenance de la destination. Le dispositif routeur était forcé d'abandonner le service.
0x1C	Missing attribute list entry data (Données d'entrée de liste d'attributs absentes)	Le service n'avait pas fourni un attribut dans la liste d'attributs qui était nécessaire au service pour avoir le comportement demandé.
0x1D	Invalid attribute value list (liste de valeurs d'attribut non valide)	Le service retourne la liste d'attributs fournie avec les informations de statut pour les attributs qui n'étaient pas valides.
0x1E	Embedded service error (Erreur de service intégré)	Un service intégré a donné lieu à une erreur.
0x1F	Vendor specific error (Erreur spécifique à un vendeur)	Une erreur spécifique à un vendeur s'est produite. Le champ code de statut étendu de la réponse d'erreur définit l'erreur particulière rencontrée. Il convient de n'utiliser ce code de statut général que si aucun des codes de statut présentés dans ce tableau ou dans la définition de classe d'objets ne reflète précisément l'erreur.
0x20	Invalid parameter (Paramètre non valide)	Un paramètre associé à la demande n'était pas valide. Ce code est utilisé lorsqu'un paramètre ne satisfait pas aux exigences de la présente spécification et/ou aux exigences définies dans une spécification d'objet d'application.
0x21	Write once value or medium already written (Une valeur ou un support inscriptible une seule fois déjà inscrite)	Une tentative est faite d'écrire sur un support inscriptible une seule fois (par exemple: lecteur non réinscriptible (WORM «Write-Once-Read-Many times»), PROM («Programmable Read Only Memory»)) qui a déjà eu une écriture ou de modifier une valeur qui ne peut pas être changée une fois établie
0x22	Invalid Response Received (Réponse non valide reçue)	Une réponse non valide est reçue (par exemple: le code de service de réponse ne concorde pas au code de service de demande ou bien le message de réponse est plus court que la taille minimale de réponse prévue). Ce code de statut peut servir pour d'autres causes de réponses non valides.
0x23	Buffer overflow (Dépassement de capacité du tampon)	Le message reçu est plus volumineux que ce que le tampon de réception peut gérer. Le message tout entier a été rejeté.
0x24	Message format error (Erreur de format de message)	Le format de message reçu n'est pas pris en charge par le serveur.
0x25	Key Failure in path (Défaillance de clé dans le chemin)	Le segment de clé qui était inclus comme premier segment dans le chemin ne concorde pas avec le module de destination. Le statut spécifique à un objet doit indiquer quelle partie de la vérification de la clé a échoué.
0x26	Path Size Invalid (Taille de chemin non valide)	Soit la taille du chemin qui était envoyé avec la demande de service n'est pas suffisamment grande pour permettre l'acheminement de la demande jusqu'à un objet, soit trop de données d'acheminement étaient incluses.
0x27	Unexpected attribute in list (Attribut inattendu dans la liste)	Une tentative a été faite de fixer un attribut qui ne pouvait pas être réglé à ce stade.
0x28	Invalid Member ID (Identificateur de membre non valide)	L'ID de membre spécifié dans la demande n'existe pas dans la Classe/l'Instance/l'Attribut spécifié(e).
0x29	Member not settable (Membre non réglable)	Une demande de modifier un membre non modifiable a été reçue.
0x2A	Group 2 only server general failure (Défaillance générale de serveur de groupe 2 seulement)	Ce code de statut peut seulement être rapporté par des serveurs CP 2/3 de Groupe 2 seulement avec un espace de code de 4 Ko ou moins et seulement à la place d'un Service non pris en charge, d'un Attribut non pris en charge et d'un Attribut non réglable.
0x2B à 0xCF	Reserved (Réservé)	Réservé pour des extensions futures

Status code	Nom	Description et signification du code de statut
0xD0 à 0xFF	Reserved for object class and service errors (Réservés pour des erreurs de service et de classe d'objets)	Cette plage de codes de statut doit être utilisée pour indiquer des erreurs spécifiques à une classe d'objets. Il convient de n'utiliser cette plage que si aucun des codes de statut présentés dans ce tableau ne reflète précisément l'erreur qui s'est produite.

NOTE La CEI 61158-5-2:2010 contient plus de détail sur les codes de statut généraux des réponses de service pour chaque service commun.

4.1.11.2.2 Format du statut étendu

Le `MR_response_Header` contient un paramètre appelé `Extended_status[]` qui est une donnée de statut étendu étiquetée.

L'utilisation et la définition réelles de ce statut étendu dépendent de la classe d'objets ou du service: chaque classe d'objets définit ses propres valeurs et plages de valeurs de statut étendu (y compris celles qui sont spécifiques à un vendeur).

4.1.11.2.3 Format des statuts général et étendu spécifiques à un objet

4.1.11.2.3.1 Généralités

Les articles ci-après spécifient le format des statuts général et étendu spécifiques à un objet

4.1.11.2.3.2 Format du statut d'objet Identity

Le Tableau 147 spécifie le format de statut général et étendu spécifiques à un objet pour l'objet Identity.

Tableau 147 – Codes de statut d'objet Identity

Code de statut général	Codes de statut étendu associés de 8 bits	Nom du statut	Description du statut
0x00 à 0xCF		Codes de statut général	Défini en 4.1.11.2.1
	0x00 à 0xEE		Codes de statut étendu réservés
	0xF0 à 0xFE	Spécifique à un vendeur	Codes de statut étendu spécifique à un vendeur
	0xFF		Utilisé avec tous les codes de statut général lorsque requis et aucun autre code de statut étendu n'est attribué.
0xD0		Hardware diagnostic (Diagnostic du matériel)	Conditions d'autotest et de diagnostic du matériel du dispositif
	0x00		Reserved
	0x01		Erreur de somme de contrôle (ou de CRC) – Espace de code/ROM – Section de boot
	0x02		Erreur de somme de contrôle (ou de CRC) – Espace de code/ROM – Section d'application
	0x03		Erreur de somme de contrôle (ou de CRC) – Mémoire NV (flash/EEPROM)
	0x04		Mémoire non volatile (NV) non valide – Mauvaise configuration

Code de statut général	Codes de statut étendu associés de 8 bits	Nom du statut	Description du statut
	0x05		Mémoire non volatile (NV) non valide – Aucune configuration établie
	0x06		Mauvaise mémoire RAM – La mémoire RAM dans le dispositif était déterminée comme comportant des cellules inopérantes.
	0x07		Mauvaise mémoire ROM/Flash
	0x08		Mauvaise mémoire Flash/EEPROM (NV)
	0x09		Erreur de câble d'interconnexion / problème de chemin de signal
	0x0A		Problème de puissance – Surintensité
	0x0B		Problème de puissance – Surtension
	0x0C		Problème de puissance – Sous-tension
	0x0D		Problème de capteur interne
	0x0E		Défaut d'horloge système
	0x0F		La configuration matérielle ne concorde pas avec la configuration NV
	0x10		Chien de garde désactivé/au repos
	0x11		Temporisateur de chien garde expiré
	0x12		Surchauffe du dispositif
	0x13		Température ambiante hors des limites de fonctionnement
	0x14 à 0xEF	(réserve)	Réserve
	0xF0 à 0xFE		Codes de statut étendu spécifique à un vendeur
	0xFF		Utilisé avec tous les codes de statut général lorsque requis et aucun autre code de statut étendu n'est attribué.
0xD1		Statut/états du dispositif	Événements et conditions de statut de dispositif
	0x01		Puissance appliquée
	0x02		Dispositif réinitialisé
	0x03		Perte de puissance de dispositif
	0x04		Activé
	0x05		Désactivé
	0x06		Entrer dans l'état d'autotest
	0x07		Entrer dans l'état d'attente
	0x08		Entrer dans l'état opérationnel
	0x09		Défaut récupérable mineur et non spécifique détecté
	0x0A		Défaut irrécupérable mineur et non spécifique détecté
	0x0B		Défaut récupérable majeur et non spécifique détecté
	0x0C		Défaut irrécupérable majeur et non spécifique détecté
	0x0D		Défaut(s) corrigé(s)
	0x0E		Modification de Ccv
	0x0F		Intervalle Heartbeat modifié
	0x10 à 0xEF	(réserve)	

Code de statut général	Codes de statut étendu associés de 8 bits	Nom du statut	Description du statut
	0xF0 à 0xFE	Spécifique à un vendeur	Spécifique à un vendeur
	0xFF		Utilisé avec tous les codes de statut général lorsque requis et aucun autre code de statut étendu n'est attribué.
0xD2 à 0xEF		Codes de statut général spécifique à un objet	Réservé – Pas encore attribué
	0x00 à 0xFF	Reserved	
0xF0 à 0xFF		Codes de statut général spécifique à un vendeur	Une erreur spécifique à un vendeur s'est produite. Le champ code de statut étendu de la réponse d'erreur définit l'erreur particulière rencontrée. Il convient de n'utiliser ce code de statut général que si aucun des codes de statut présentés dans ce tableau ou dans la définition de classe d'objets ne reflète précisément l'erreur.
	0x00 à 0xFF	Codes de statut étendu spécifique à un vendeur	Tous les codes de statut étendu sont disponibles pour une association avec chaque code de statut général spécifique à un vendeur.

4.2 Spécification de syntaxe abstraite des données

4.2.1 Spécification de format de transport

Les couches inférieures des architectures de systèmes ouverts sont concernées par le transport de données utilisateur parmi les unités fonctionnelles distribuées. Dans ces couches, les données utilisateur peuvent être considérées simplement comme une séquence d'octets. Cependant, des entités de couche application peuvent manipuler les valeurs de types de données tout à fait complexes. Pour assurer l'indépendance entre la couche application et les couches inférieures, les types de données peuvent être spécifiés dans une notation de syntaxe abstraite.

Le fait de compléter la syntaxe abstraite par un ou plusieurs algorithmes (appelés règles de codage) peut déterminer les valeurs des octets de couche inférieure qui transportent les valeurs de couche application. La combinaison de la syntaxe abstraite avec un seul jeu de règles de transfert produit une syntaxe de transfert spécifique.

4.2.2 Notation de syntaxe abstraite

Les définitions de types de données fournies dans la présente NORME doivent être écrites en Notation de syntaxe abstraite numéro un (ASN.1), telle que définie dans la série ISO/CEI 8824. Ces définitions de types doivent faire partie du module "Network DataTypes" de l'ASN.1. La déclaration de début ASN.1 indiquant que ces définitions se trouvent dans ce module est:

```
control network DataTypes DEFINITIONS ::= BEGIN
```

et la déclaration de fermeture ASN.1 doit être le mot-clé "END".

Les définitions abstraites qui suivent doivent comprendre le jeu de types de données réseau de commande. En outre, une disposition est prise pour étendre ou dériver de nouveaux types de données basés sur des types définis existants et pour les inclure dans un "dictionnaire de types".

4.2.3 Spécification des données réseau de commande

La notation [typeId] pour les données de chaîne de bits structurées et de sous-plage énumérées directement dérivées doit signifier que l'étiquette doit être prise dans le champ "type" de la VariableDictionaryEntry correspondante.

```

Network Data ::= CHOICE{ElementaryData, DerivedData}
ElementaryData ::= CHOICE{
    BOOL,
    FixedLengthInteger,
    FixedLengthReal,
    AnyTime,
    AnyDate,
    AnyString,
    FixedLengthBitString,
    EPATH}
DerivedData ::= CHOICE {
    DirectlyDerivedData,
    EnumeratedData,
    SubrangeData,
    StructuredBitStringData,
    ARRAY,
    STRUCT,
    FunctionBlockData}
DirectlyDerivedData ::= [typeId] NetworkData
EnumeratedData ::= [typeId] USINT
SubrangeData ::= [typeId] FixedLengthInteger
StructuredBitStringData ::= [typeId] FixedLengthBitString
FixedLengthInteger ::= CHOICE {SignedInteger, UnsignedInteger}
SignedInteger ::= CHOICE {SINT, INT, DINT, LINT}
UnsignedInteger ::= CHOICE {USINT, UINT, UDINT, ULINT}
FixedLengthReal ::= CHOICE {REAL, LREAL}
AnyTime ::= CHOICE {ITIME, TIME, FTIME, LTIME}
AnyDate ::= CHOICE {DATE, TIME_OF_DAY, DATE_AND_TIME}
AnyString ::= CHOICE {STRING, STRING2}
FixedLengthBitString ::= CHOICE {SWORD, WORD, DWORD, LWORD}
BOOL ::= [PRIVATE 1] IMPLICIT BOOLEAN
SINT ::= [PRIVATE 2] IMPLICIT OCTET STRING-- 1 octet
INT ::= [PRIVATE 3] IMPLICIT OCTET STRING-- 2 octets
DINT ::= [PRIVATE 4] IMPLICIT OCTET STRING-- 4 octets
LINT ::= [PRIVATE 5] IMPLICIT OCTET STRING-- 8 octets
USINT ::= [PRIVATE 6] IMPLICIT OCTET STRING-- 1 octet
UINT ::= [PRIVATE 7] IMPLICIT OCTET STRING-- 2 octets
UDINT ::= [PRIVATE 8] IMPLICIT OCTET STRING-- 4 octets
ULINT ::= [PRIVATE 9] IMPLICIT OCTET STRING-- 8 octets
REAL ::= [PRIVATE 10] IMPLICIT OCTET STRING-- 4 octets
LREAL ::= [PRIVATE 11] IMPLICIT OCTET STRING-- 8 octets
STIME ::= [PRIVATE 12] IMPLICIT DINT

```

```

DATE ::= [PRIVATE 13] IMPLICIT UINT
TIME_OF_DAY ::= [PRIVATE 14] IMPLICIT UDINT
DATE_AND_TIME ::= [PRIVATE 15] IMPLICIT SEQUENCE {
    time_of_day UDINT,
    date UINT }
STRING ::= [PRIVATE 16] IMPLICIT SEQUENCE {
    charcount      UINT,
    stringcontents OCTET STRING} -- one octet per character
SWORD ::= [PRIVATE 17] IMPLICIT OCTET STRING-- 1 octet
WORD ::= [PRIVATE 18] IMPLICIT OCTET STRING-- 2 octets
DWORD ::= [PRIVATE 19] IMPLICIT OCTET STRING-- 4 octets
LWORD ::= [PRIVATE 20] IMPLICIT OCTET STRING-- 8 octets
STRING2 ::= [PRIVATE 21] IMPLICIT SEQUENCE {
    charcount      UINT,
    string2contents OCTET STRING} -- 2 octets/ character
FTIME ::= [PRIVATE 22] IMPLICIT DINT
LTIME ::= [PRIVATE 23] IMPLICIT LINT
ITIME ::= [PRIVATE 24] IMPLICIT INT
STRINGN ::= [PRIVATE 25] IMPLICIT SEQUENCE {
    charsize      UINT,
    charcount      UINT,
    stringNcontents OCTET STRING} -- N octets/ character
SHORT_STRING ::= [PRIVATE 26] IMPLICIT SEQUENCE {
    charcount      USINT,
    stringcontents OCTET STRING} -- one octet per character
TIME ::= [PRIVATE 27] IMPLICIT DINT
EPATH ::= [PRIVATE 28] IMPLICIT OCTET STRING -- IEC 61158-6-2
STRINGI ::= [PRIVATE 30] IMPLICIT SEQUENCE{
    stringnum USINT (number of strings)
    array of STRUCT
        language1 USINT (first character from ISO 639-2/T)
        language2 USINT (second character from ISO 639-2/T)
        language3 USINT (third character from ISO 639-2/T)
        datatype EPATH limited to the values 0xD0,0xD5,0xD9,and
0xDA)
        charset      UINT      from IANA MIB Printer Codes
(IETF RFC 1759))
    stringcontents CHOICE OF (SHORT_STRING, STRING, STRING2,
or STRINGN) -- based on datatype field
ARRAY ::= SEQUENCE OF NetworkData -- All of same base type
STRUCT ::= SEQUENCE OF NetworkData -- May be different types
FunctionBlockData ::= SET{
    inputs [0] IMPLICIT STRUCT OPTIONAL,
    outputs [1] IMPLICIT STRUCT OPTIONAL,
    controlInputs [2] IMPLICIT STRUCT OPTIONAL,
    controlOutputs [3] IMPLICIT STRUCT OPTIONAL}

```

4.2.4 Spécification/dictionnaires de types de données

La définition d'un objet peut inclure du texte qui définit des attributs. Les attributs doivent recevoir un *Data Type* dans une spécification d'objet. Le Data Type peut être l'un de ceux

définis dans la présente norme ou peuvent être une extension spécifique à un objet apportée à la présente norme. La définition doit fournir une spécification de type (*Type Specification*) pour les données et doit fournir une structure pour étendre ou dériver de nouveaux types de données sur la base de types définis existants.

```
Dictionary ::= CHOICE {VariableDictionary, TypeDictionary}
VariableDictionary ::= SEQUENCE OF VariableDictionaryEntry
VariableDictionaryEntry ::= SEQUENCE{
    name AnyString,
    id FixedLengthInteger,
    type TypeID,
    ranges SEQUENCE OF Subrange, -- for arrays
    accessPrivilege BOOL {READ_ONLY(0), READ_WRITE(1)}
TypeID ::= OCTET STRING -- ASN.1 encoded tag value of the
-- DataTypeSpecification module
Subrange ::= SEQUENCE {
    minValue FixedLengthInteger,
    maxValue FixedLengthInteger}
TypeDictionary ::= SEQUENCE OF TypeDictionaryEntry
TypeDictionaryEntry ::= SEQUENCE {
    name AnyString,
    type TypeID,
    spec DataTypeSpecification}
```

```

DataTypeSpecification ::= CHOICE {
    alt      [PRIVATE 0] IMPLICIT AlternateTypeSpec,
    bool     [PRIVATE 1] IMPLICIT NULL, -- BOOL
    sint     [PRIVATE 2] IMPLICIT NULL, -- SINT
    int      [PRIVATE 3] IMPLICIT NULL, -- INT
    dint     [PRIVATE 4] IMPLICIT NULL, -- DINT
    lint     [PRIVATE 5] IMPLICIT NULL, -- LINT
    usint    [PRIVATE 6] IMPLICIT NULL, -- USINT
    uint     [PRIVATE 7] IMPLICIT NULL, -- UINT
    udint    [PRIVATE 8] IMPLICIT NULL, -- UDINT
    ulint    [PRIVATE 9] IMPLICIT NULL, -- ULINT
    real     [PRIVATE 10] IMPLICIT NULL, -- REAL
    lreal    [PRIVATE 11] IMPLICIT NULL, -- LREAL
    stime    [PRIVATE 12] IMPLICIT NULL, -- STIME
    date     [PRIVATE 13] IMPLICIT NULL, -- DATE
    tod      [PRIVATE 14] IMPLICIT NULL, -- TIME_OF_DAY
    dat      [PRIVATE 15] IMPLICIT NULL, -- DATE_AND_TIME
    str1     [PRIVATE 16] IMPLICIT NULL, -- STRING
    sword    [PRIVATE 17] IMPLICIT NULL, -- SWORD
    word     [PRIVATE 18] IMPLICIT NULL, -- WORD
    dword    [PRIVATE 19] IMPLICIT NULL, -- DWORD
    lword    [PRIVATE 20] IMPLICIT NULL, -- LWORD
    str2     [PRIVATE 21] IMPLICIT NULL, -- STRING2
    ftime    [PRIVATE 22] IMPLICIT NULL, -- FTIME
    ltime    [PRIVATE 23] IMPLICIT NULL, -- LTIME
    itime    [PRIVATE 24] IMPLICIT NULL, -- ITIME
    strN     [PRIVATE 25] IMPLICIT NULL, -- STRINGN
    shstr    [PRIVATE 26] IMPLICIT NULL, -- SHORT_STRING
    time     [PRIVATE 27] IMPLICIT NULL, -- TIME
    epath    [PRIVATE 28] IMPLICIT NULL, -- EPATH
    strI     [PRIVATE 30] IMPLICIT NULL, -- STRINGI
    constructedData CHOICE {
        abbrevStruc [0] IMPLICIT AbbreviatedStrucTypeSpec,
        abbrevArr  [1] IMPLICIT AbbreviatedArrayTypeSpec,
        frmlStruc  [2] IMPLICIT FormalStrucTypeSpec,
        frmlArr    [3] IMPLICIT FormalArrayTypeSpec,
        expBitStr  [4] IMPLICIT ExpandedFixedLenBitStrTypeSpec,
        expStr1    [5] IMPLICIT ExpandedStringTypeSpec,
        expStr2    [6] IMPLICIT ExpandedString2TypeSpec
    }
}

AbbreviatedStrucTypeSpec ::= UINT
AbbreviatedArrayTypeSpec ::= DataTypeSpecification
FormalStrucTypeSpec ::= SEQUENCE OF DataTypeSpecification
FormalArrayTypeSpec ::= SEQUENCE {
    lowBound [0] IMPLICIT FixedLengthInteger, -- Array Lower Bound
    highBound [1] IMPLICIT FixedLengthInteger, -- Array Upper Bound
    dataType  DataTypeSpecification
}

```

```

ExpandedFixedLenBitStrTypeSpec ::= SEQUENCE {
    bitStrType DataTypeSpecification    -- SWORD, WORD, DWORD, or
    LWORD
    bitFields [7] IMPLICIT BitFieldDef}

BitFieldDef ::= SEQUENCE OF {
    bitDef [2] IMPLICIT OCTET STRING}    -- Length is always 2
octets.
                                     -- First octet contains starting
                                     -- Bit Position. Trailing octet
                                     -- contains the number of bits.

ExpandedStringTypeSpec ::= UINT-- String Length In Octets
ExpandedString2TypeSpec ::= UINT-- String Length In Octets
AlternateTypeSpec ::= CHOICE {
    directlyDerivedTypeSpec [0]    IMPLICIT TypeID,
    subrangeTypeSpec            [1]    IMPLICIT SubrangeTypeSpec ,
    enumeratedTypeSpec          [2]    IMPLICIT EnumeratedTypeSpec,
    fbTypeSpec                  [3]    IMPLICIT FBTypeSpec}

SubrangeTypeSpec ::= SEQUENCE{
    baseType    TypeID,    -- NOTE minVale and maxVale
    minValue    FixedLengthInteger, -- shall be within the range
    maxValue    FixedLengthInteger} -- of baseType values

EnumeratedTypeSpec ::= SEQUENCE OF AnyString
BitNameDefintion ::= SEQUENCE {
    bitName    AnyString,
    bitNumber    USINT}
FBTypeSpec ::= SET{
    inputs        [0] IMPLICIT FbtElementSpec OPTIONAL,
    outputs       [1] IMPLICIT FbtElementTypeSpec OPTIONAL,
    controlInputs [2] IMPLICIT FbtElementTypeSpec OPTIONAL,
    controlOutputs [3] IMPLICIT FbtElementTypeSpec OPTIONAL}

FbtElementTypeSpec ::= SEQUENCE OF ElementSpec
ElementSpec ::= SEQUENCE {
    name    AnyString,
    typespec    ElementTypeSpec}

ElementTypeSpec ::= CHOICE {
    [0] IMPLICIT TypeID,
    [1] IMPLICIT SubrangeTypeSpec,
    [2] IMPLICIT EnumeratedTypeSpec,
    [3] IMPLICIT FormalArrayTypeSpec,
    [4] IMPLICIT ExpandedStringTypeSpec,
    [5] IMPLICIT ExpandedString2TypeSpec}

```

La déclaration END qui suit doit terminer le module ASN.1 ouvert en 4.1.

END.

4.3 Syntaxe abstraite d'encapsulation

4.3.1 Protocole d'encapsulation

4.3.1.1 Note communes

NOTE Afin d'envoyer des TPDU sur TCP/IP, un protocole d'encapsulation est requis. Le présent paragraphe définit les exigences relatives au protocole d'encapsulation. Le protocole d'encapsulation est un protocole générique qui peut être utilisé pour transporter des données autres que des TPDU de Type 2.

4.3.1.2 Messages d'encapsulation

Tous les messages d'encapsulation doivent être constitués d'un en-tête de longueur fixe de 24 octets suivi d'une partie "données" facultative. La longueur totale des messages d'encapsulation (en-tête compris) doit être limitée à 65 535 octets. La longueur des messages d'encapsulation ne doit pas neutraliser les restrictions de longueur imposées par le protocole encapsulé. Sa structure doit être telle que montrée à la Figure 13, où le haut du diagramme indique la partie du message envoyé en premier sur la ligne.

NOTE La partie "données" d'encapsulation du message est facultative.

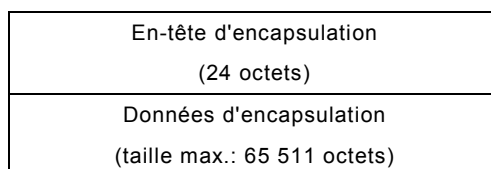


Figure 13 – Message d'encapsulation

4.3.1.3 En-tête d'encapsulation

L'en-tête d'encapsulation doit être tel que montré dans le Tableau 148.

Tableau 148 – En-tête d'encapsulation

Nom de champ	Format	Description
Command	UINT	Commande d'encapsulation
Length	UINT	Longueur, en octets, de la partie "données" du message, c'est-à-dire le nombre d'octets suivant l'en-tête
Session handle	UDINT	Identification de session (dépend de l'application)
Status	UDINT	Code de statut
Sender context	ARRAY of 8 USHORT	Informations pertinentes seulement pour l'expéditeur d'une commande d'encapsulation
Options	UDINT	Flags d'options

Les champs entiers multioctets dans les messages d'encapsulation doivent être émis comme spécifié à l'Article 5.

Bien que l'en-tête ne contienne aucune information explicite pour distinguer entre une demande et une réponse, ces informations doivent être déterminées de l'une de deux façons:

- de façon implicite, par la commande et le contexte dans lequel le message est généré. (Par exemple, dans le cas de la commande RegisterSession, la demande est générée par un émetteur et la cible génère la réponse);
- de façon explicite, par le contenu d'un paquet de protocole encapsulé dans la partie "données" du message.

4.3.1.4 Champ "Command"

L'allocation de codes de commande doit être telle que montrée dans le Tableau 149.

Tableau 149 – Codes de commande d'encapsulation

Code de "Command"	Command
0x00	Nop
0x01 à 0x03	Reserved
0x04	ListServices
0x05 à 0x64	Reserved
0x65	ResgisterSession
0x66	UnRegisterSession
0x67 à 0x6E	Reserved
0x6F	SendRRData
0x70	SendUnitData
0x71 à 0xFFFF	Reserved

Un dispositif doit accepter des commandes qu'il prend effectivement en charge sans interrompre la session ou la connexion TCP sous-jacente. Un code de statut indiquant qu'une commande non prise en charge a été reçue doit être retourné à l'expéditeur du message (voir 4.3.1.7).

4.3.1.5 Champ "Length"

Le champ "Length" (longueur) dans l'en-tête doit spécifier la taille en octets de la partie "données" du message. Le champ doit contenir zéro pour les messages qui ne contiennent pas de données. La longueur totale d'un message doit être la somme du nombre contenu dans le champ Length plus la taille de 24 octets de l'en-tête d'encapsulation.

Le message d'encapsulation complet doit être lu même si la longueur peut ne pas être valide pour une commande particulière ou peut dépasser la capacité des tampons internes de la cible.

NOTE L'échec à lire le message complet peut se traduire par la perte de trace des frontières du message dans le train d'octets TCP.

4.3.1.6 Champ "Session handle"

Le champ "Session Handle" (identification de session) doit être généré par la cible et retourné à l'émetteur en réponse à une demande RegisterSession. L'émetteur doit l'insérer dans toutes les demandes de session ultérieures (demandes utilisant des commandes de session) à cette cible particulière. Si c'est la cible qui déclenche et envoie une commande à l'émetteur, la cible doit inclure ce champ dans la demande qu'elle envoie à l'émetteur.

NOTE Certaines commandes (c'est-à-dire Nop) n'exigent pas d'identification de session, même si une session a été établie. La réponse à la question de savoir si, oui ou non, une commande particulière exige une session est notée dans la description de la commande en question.

4.3.1.7 Champ "Status"

La valeur dans le champ Status (statut) indique si, oui ou non, le destinataire était capable d'exécuter la commande d'encapsulation demandée. Une valeur de zéro dans une réponse doit indiquer une exécution réussie de la commande. Dans toutes les demandes émises par l'expéditeur, le champ Status doit contenir zéro. Si le destinataire reçoit une demande avec un champ Status différent de zéro, la demande doit être ignorée et aucune réponse ne doit être générée.

NOTE Ce champ ne reflète pas les erreurs qui sont générées par un paquet de protocole encapsulé contenu dans la partie "données" du message. Par exemple, une erreur rencontrée au cours du traitement d'un nœud final d'un service d'établissement d'attributs (Set Attributes).

Les codes de statut doivent être tels que montrés dans le Tableau 150.

Tableau 150 – Codes de statut d'encapsulation

Code de statut	Description
0x00	Succès
0x01	L'expéditeur a émis une commande d'encapsulation qui n'est pas valide ou n'est pas prise en charge.
0x02	Ressources mémoire insuffisantes dans le récepteur pour gérer la commande.
0x03	Données formées médiocrement ou incorrectes dans la partie "données" du message d'encapsulation.
0x04 à 0x63	Reserved
0x64	Un émetteur a utilisé une identification de session invalide pour envoyer un message d'encapsulation à la cible.
0x65	La cible a reçu un message dont la longueur n'est pas valide (voir 4.3.1.5).
0x66 à 0x68	Reserved
0x69	Révision de protocole non prise en charge
0x6A à 0xFFFF	Réservés

4.3.1.8 Champ "Sender context"

L'expéditeur de la commande doit attribuer la valeur dans le champ Sender Context (contexte d'expéditeur) de l'en-tête. Le destinataire doit retourner cette valeur sans sa réponse.

4.3.1.9 Champ "Options"

Pour encapsuler des PDU issues de la couche Réseau et Transport, les fanions Options doivent être mis à zéro. L'allocation des fanions Options doit être telle que montrée dans le Tableau 151.

Tableau 151 – Fanions d'options

Fanions d'options	Définition
Bits 0 à 15	Alloués pour la compatibilité avec des protocoles existants
Bits 16 à 31	Zone pour extension future

4.3.2 Descriptions des commandes

4.3.2.1 Nop

Un émetteur ou bien une cible peut envoyer une demande Nop Aucune réponse ne doit être générée. La partie "données" de la demande doit avoir une longueur de 0 octet à 65 511 octets. Le destinataire doit ignorer toute donnée contenue dans le message.

NOTE Une commande Nop fournit une façon pour un émetteur ou bien une cible de déterminer si la connexion TCP est encore ouverte.

L'en-tête d'encapsulation de la demande Nop doit être tel que montré dans le Tableau 152.

Tableau 152 – En-tête d'encapsulation de demande Nop

Nom de champ	Valeur du champ
Command	Nop (0)
Length	Longueur de la partie "données"
Session handle	Ignoré
Status	0
Sender context	Ignoré
Options	0

4.3.2.2 RegisterSession

Un émetteur doit envoyer une demande RegisterSession à une cible pour lancer une session.

NOTE Voir 11.8 pour des informations détaillées sur l'établissement et le maintien d'une session.

L'en-tête d'encapsulation de la demande RegisterSession doit être tel que montré dans le Tableau 153.

Tableau 153 – En-tête d'encapsulation de demande RegisterSession

Nom de champ	Valeur du champ
Command	RegisterSession (0x65)
Length	4
Session handle	0
Status	0
Sender context	Sender context
Options	0

Les paramètres dans la partie "données" doit déterminer la version du protocole et les éventuelles options de session comme montré dans le Tableau 154. La version de protocole doit être mise à 1. Les options de session doivent être mises à 0.

Tableau 154 – Partie "données" de la demande RegisterSession

Champ	Type	Description
Protocol version	UINT	Version de protocole demandée (1)
Options flags	UINT	Options de session (0)

Le Tableau 155 ci-dessous montre l'organisation des fanions Options:

Tableau 155 – Fanions d'options

Fanions d'options	Définition
Bits 0 à 7	Alloués pour la compatibilité avec des protocoles existants
Bits 8 à 15	Réservé pour extension future

La cible envoie une réponse RegisterSession pour indiquer qu'il a enregistré l'émetteur. La réponse doit avoir le même format que la demande, comme montré dans le Tableau 156.

Tableau 156 – En-tête d'encapsulation de réponse RegisterSession

Nom de champ	Valeur du champ
Command	RegisterSession (0x65)
Length	4
Session handle	Identification retournée par la cible
Status	0
Sender context	Contexte préservé issu de la demande
Options	0

Le champ Session Handle de l'en-tête doit contenir un identificateur généré par la cible que l'émetteur doit sauvegarder et insérer dans le champ Session Handle de l'en-tête de toutes les demandes ultérieures envoyées à la cible en question. Ce champ doit être valide seulement si le champ Status est zéro (0).

Le champ Sender Context de l'en-tête doit contenir les mêmes valeurs présentes dans la demande de l'expéditeur d'origine. Si l'émetteur a été enregistré auprès de la cible, le champ Status doit être zéro (0). Si la cible a été incapable d'enregistrer, le champ Status doit être différent de zéro.

La partie "données" de la réponse doit avoir le même format que la demande, comme montré dans le Tableau 157.

Tableau 157 – Partie "données" de la réponse RegisterSession

Champ	Type	Description
Protocol Version	UINT	Version de protocole prise en charge
Options	UINT	Options de session préservées issues de la demande

Le champ Protocol Version doit être égal à la version demandée si l'émetteur a été enregistré avec succès. Si la cible ne prend pas en charge la version demandée du protocole:

- la session ne doit pas être créée;
- le champ Status doit être mis;
- la cible doit retourner la version prise en charge la plus élevée dans le champ Protocol Version.

Si toutes les options demandées sont prises en charge, le champ Options doit retourner la valeur de l'émetteur. Cette valeur doit être zéro.

4.3.2.3 UnRegisterSession

Un émetteur ou bien une cible peut envoyer cette demande de mettre fin à la session. Le destinataire doit lancer une fermeture de la connexion TCP/IP sous-jacente lorsqu'il reçoit cette demande. La session doit aussi être arrêtée lorsqu'il est mis fin à la connexion de transport entre l'émetteur et la cible. Le destinataire doit accomplir toute mise au net associée exigée sur son côté. Il ne doit pas y avoir de réponse à cette commande.

Le format de la demande UnregisterSession doit être tel que montré dans le Tableau 158.

Tableau 158 – En-tête d'encapsulation de demande UnRegisterSession

Nom de champ	Valeur du champ
Command	UnRegisterSession (0x65)
Length	4
Session Handle	Identification provenant de RegisterSession
Status	0
Sender Context	Contexte de l'expéditeur
Options	0

Le champ Session Handle doit être mis à la valeur obtenue par la réponse RegisterSession d'origine. L'identification doit devenir non valide après que cette demande a aura été reçue et traitée par la cible.

4.3.2.4 ListServices

La demande ListServices doit être envoyée pour déterminer quelles classes de services d'encapsulation le dispositif-cible prend en charge.

NOTE Chaque classe de services a un code de type unique et un nom ASCII facultatif.

Une seule classe de services est définie, avec le code de type 0x100 et le nom "Communications". Cette classe de service doit indiquer que le dispositif prend en charge l'encapsulation des PDU de Type 2. Tous les dispositifs qui prennent en charge l'encapsulation des PDU de Type 2 doivent prendre en charge la commande ListServices et la classe de services "Communications".

L'en-tête d'encapsulation de la demande ListServices doit être tel que montré dans le Tableau 159.

Tableau 159 – En-tête d'encapsulation de demande ListServices

Nom de champ	Valeur du champ
Command	ListServices (0x04)
Length	0
Session handle	Ignoré
Status	0
Sender context	Choisi par l'expéditeur
Options	0

Le destinataire doit répondre par un message d'encapsulation constitué de l'en-tête et de données, comme montré dans le Tableau 160 et dans le Tableau 161. La partie "données" de la réponse doit fournir des informations sur les services pris en charge.

Tableau 160 – En-tête d'encapsulation de réponse ListServices

Nom de champ	Valeur du champ
Command	ListServices (0x04)
Length	0
Session handle	Ignoré
Status	0
Sender context	Préservé issu de la demande
Options	0

La partie "données" de la réponse doit contenir un compte d'objets de 2 octets suivi d'une matrice d'objets décrivant le(s) service(s) fournis, comme montré dans le Tableau 161.

Tableau 161 – Partie "données" de la réponse ListServices

Nom de champ	Type	Description
Object count	UINT	Nombre d'objets de services à suivre
Type code	UINT	Code de type de service
Object length	UINT	Longueur restante en octets
Version	UINT	Version de protocole d'encapsulation
Flags	UINT	Fanions de capacité
Name	ARRAY of 16 USINT	Nom de service

Le code de type (Type Code) doit identifier la classe de services, comme montré dans le Tableau 162.

Tableau 162 – Codes de type de services

Code de type de service	Description
0x00 à 0xFF	Reserved
0x100	Service Communications
0x101 à 0xFFFF	Réservés

Le champ Version doit indiquer la version du service prise en charge par la cible pour aider à maintenir la compatibilité entre les applications.

Chaque service doit avoir un jeu différent de fanions de capacité. Les fanions non utilisés doivent être mis à zéro.

Les fanions de capacité, définis pour le service Communications, doivent être tels que montrés dans le Tableau 163.

Tableau 163 – Fanions de capacité de communications

Valeur du fanion	Description
Bits 0 à 4	Réservés
Bit 5	Si le dispositif prend en charge l'encapsulation des PDU de Type 2, ce bit doit être mis (= 1); autrement, il doit être effacé (= 0).
Bits 6 à 7	Réservés
Bit 8	Prend en charge les connexions de Classe 0 et 1 basées sur UDP
Bits 9 à 15	Réservés

Le champ Name doit permettre une chaîne ASCII terminée par le caractère NULL pouvant atteindre 16 octets au maximum à des fins de description uniquement. La limite de 16 octets doit inclure le caractère NULL.

4.3.2.5 ListIdentity

Un émetteur de connexion peut utiliser la demande ListIdentity pour localiser et identifier des cibles potentielles. Cette demande doit être envoyée comme un message de diffusion utilisant le protocole UDP et ne doit pas exiger qu'une session soit établie.

L'en-tête d'encapsulation de la demande ListIdentity doit être tel que montré dans le Tableau 164.

Tableau 164 – En-tête d'encapsulation de demande ListIdentity

Nom de champ	Valeur du champ
Command	ListIdentity (0x63)
Length	0
Session handle	Ce champ est ignoré, car une session peut ne pas être établie avant d'envoyer la demande ListIdentity.
Status	0
Sender context	0
Options	0

Un seul élément de réponse est défini pour cette commande, Target Identity (identité de cible), avec le code de type d'élément 0x0C. Cet élément doit être pris en charge (retourné) par tous les dispositifs capables du Type 2

Chaque destinataire de la commande ListIdentity doit répondre par un message d'encapsulation constitué de l'en-tête et de données, comme montré dans le Tableau 165 et dans le Tableau 166. La partie "données" du message doit fournir des informations sur l'identité des cibles. La réponse doit être envoyée à l'adresse IP en provenance de laquelle la demande de diffusion a été reçue.

Tableau 165 – En-tête d'encapsulation de réponse ListIdentity

Nom de champ	Valeur du champ
Command	List Identity (0x63)
Length	0
Session handle	Ignoré
Status	0
Sender context	0
Options	0

La partie "données" de la réponse est structurée comme un Common Packet Format (format commun de paquet) qui contient un compte d'objets de 2 octets suivi d'une matrice d'objets fournissant l'identité de la cible, comme montré dans le Tableau 166.

À tout le moins, l'élément Identity CFP 2 doit être retourné et a le format tel que défini dans le Tableau 166. Une partie de cette définition d'élément suit la définition de réponse de service Get Attribute All de l'objet Identity (données retournées sur la base de l'instance une de cet objet) et peut adopter plusieurs membres dans la mesure où de nouveaux membres sont ajoutés à la réponse de service en question. Contrairement à la plupart des champs dans le format commun de paquet, le champ Socket Address doit être envoyé dans un ordre grossier.

Tableau 166 – Partie "données" de la réponse ListIdentity

Nom de champ	Type	Description
Object count	UINT	Nombre d'objets Identity à suivre
Type code	UINT	Code indiquant le type d'objet Identity CPF 2 (0x0C)
Object length	UINT	Longueur restante en octets
Version	UINT	Version de protocole d'encapsulation
Socket Address	ARRAY of:	Adresse de port de connexion
	INT	sin_family (gros boutiste)
	UINT	sin_port (gros-boutiste)
	UDINT	sin_addr (gros-boutiste)
	ARRAY of USINT	sin_zero (longueur de 8) (gros-boutiste)
Vendor ID ^a	UINT	ID de vendeur des fabricants de dispositifs
Device Type ^a	UINT	Type de dispositif du produit
Product Code ^a	UINT	Code de produit attribué par rapport au type de dispositif
Revision ^a	USINT[2]	Révision du dispositif
Status ^a	WORD	Statut actuel du dispositif
Serial Number ^a	UDINT	Numéro de série du dispositif
Product Name ^a	SHORT_STRING	Description du dispositif lisible par l'homme
State ^a	USINT	État actuel du dispositif
^a Ces paramètres sont définis en plus par l'attribut d'instance correspondant de l'objet Identity.		

4.3.2.6 ListInterfaces

La demande facultative ListInterfaces doit être utilisée par un émetteur de connexion pour identifier des interfaces potentielles de communications autres que de Type 2 associées à la cible. Une session peut ne pas être établie pour envoyer cette commande.

L'en-tête d'encapsulation de la demande ListInterfaces doit être tel que montré dans le Tableau 167.

Tableau 167 – En-tête d'encapsulation de demande ListInterfaces

Nom de champ	Valeur du champ
Command	List Interfaces (0x64)
Length	0
Session handle	Ignoré
Status	0
Sender context	0
Options	0

Si elle est prise en charge, le destinataire d'une commande de demande ListInterfaces doit répondre par un message d'encapsulation constitué de l'en-tête et de données, comme montré dans le

Tableau 168.

Tableau 168 – En-tête d'encapsulation de réponse ListInterfaces

Nom de champ	Valeur du champ
Command	List Interfaces (0x64)
Length	0
Session handle	Ignoré
Status	0
Sender context	0
Options	0

La partie "données" du message est structurée comme un Common Packet Format (format commun de paquet) qui contient un compte d'objets de 2 octets suivi d'une matrice d'objets fournissant des informations relatives aux interfaces de communications autres que de Type 2 associées à la cible.

Il n'y a pas d'éléments définis publiquement qui soient retournés avec cette réponse. L'/Les élément(s) spécifique(s) à un vendeur qui est/sont retourné(s) doi(ven)t, à tout le moins, retourner une Interface Handle (identification d'interface) de 32 bits qui est utilisée par d'autres commandes d'encapsulation, par exemple, la commande SendRRData.

4.3.2.7 SendRRData

Une demande SendRRData doit transférer un paquet encapsulé de demande/réponse entre l'émetteur et la cible, l'émetteur lançant la commande. Les paquets réels de demande/réponse doivent être encapsulés dans la partie "données" du message et doivent être de la responsabilité de la cible et de l'émetteur. La commande SendRRData doit être utilisée pour envoyer des messages UCMM encapsulés.

L'en-tête d'encapsulation de la demande SendRRData doit être tel que montré dans le Tableau 169.

Tableau 169 – En-tête d'encapsulation de demande SendRRData

Nom de champ	Valeur du champ
Command	SendRRData (0x6F)
Length	Longueur de la partie "données"
Session handle	Identification retournée par RegisterSession
Status	0
Sender context	Contexte de l'expéditeur
Options	0

La partie "données" du message doit contenir les paramètres de demande et le paquet de protocole encapsulé, comme montré dans le Tableau 170.

Tableau 170 – Partie "données" de la demande SendRRData

Nom de champ	Type	Description
Interface handle	UDINT	Doit être 0
Timeout	UINT	Temporisation d'opération
Paquet de protocole encapsulé (voir la spécification de Common Packet Format en 4.3.3)		

Le champ Interface handle doit identifier l'interface de communications à laquelle la demande est dirigée Cette identification doit être 0 pour les PDU d'encapsulation de Type 2.

La cible doit abandonner l'opération demandée après que la temporisation expire. Les valeurs de temporisation doivent être exprimées en secondes. La valeur minimale de temporisation doit être une (1) seconde; la valeur maximale doit être 65 535 s (environ 18 h). Une valeur de zéro doit indiquer que l'opération doit seulement être abandonnée lorsque le protocole encapsulé temporise. Comme la couche Réseau et Transport définit déjà des mécanismes de temporisation pour les messages connectés et non connectés, le champ Timeout doit être zéro.

Le paquet de protocole encapsulé doit être codé dans un Common Packet Format, comme montré dans le 4.3.3.

La réponse SendRRData, telle que montrée dans le Tableau 171, doit contenir des données en réponse à la demande SendRRData. La réponse à la demande de protocole encapsulé d'origine doit être contenue dans la partie "données" de la réponse SendRRData.

Tableau 171 – En-tête d'encapsulation de réponse SendRRData

Nom de champ	Valeur du champ
Command	SendRRData (0x6F)
Length	Longueur de la partie "données"
Session handle	Identification retournée par RegisterSession
Status	0
Sender context	Contexte d'expéditeur préservé
Options	0

Le format de la partie "données" du message de réponse doit être le même que celui du message de demande SendRRData.

4.3.2.8 SendUnitData

La demande SendUnitData doit envoyer des messages connectés encapsulés. Une demande peut ne pas être retournée. Cette commande peut être utilisée lorsque le protocole encapsulé a son propre mécanisme sous-jacent de transport de bout en bout.

Le format de la demande SendUnitData doit être tel que montré dans le Tableau 172.

Tableau 172 – En-tête d'encapsulation de demande SendUnitData

Nom de champ	Valeur du champ
Command	SendUnitData (0x70)
Length	Longueur de la partie "données"
Session handle	Identification retournée par RegisterSession
Status	0
Sender context	Contexte de l'expéditeur
Options	0

La partie "données" du message doit contenir les paramètres de demande ainsi que le paquet de protocole encapsulé, comme montré dans le Tableau 173.

Tableau 173 – Partie "données" de la demande SendUnitData

Nom de champ	Type	Description
Interface handle	UDINT	Doit être 0.
Timeout	UINT	Temporisation d'opération
Paquet de protocole encapsulé (voir la spécification de Common Packet Format en 4.3.3)		

Les champs Interface handle et Timeout doivent être les mêmes que dans la demande SendRRData.

4.3.3 Common Packet Format (Format commun des paquets)

4.3.3.1 Généralités

Le format commun des paquets doit être constitué d'un compte d'éléments, suivi d'un élément d'adresse puis d'un élément de donnée (dans cet ordre), comme montré dans le Tableau 174. D'autres éléments facultatifs peuvent suivre.

NOTE Le format commun des paquets définit un format normalisé pour les paquets de protocole qui sont transportés avec le protocole d'encapsulation. Le format commun des paquets est un mécanisme d'usage général conçu pour prendre en charge les futurs types de paquet ou d'adresse.

Tableau 174 – Format commun des paquets

Nom de champ	Type	Description
Item count	UINT	Nombre d'éléments à suivre (doit être au moins égal à 2)
Address item	Item Struct (voir ci-dessous)	Informations d'adressage pour paquet encapsulé
Data item	Item Struct (voir ci-dessous)	Le paquet de données encapsulé
Éléments supplémentaires facultatifs.		

La structure des éléments adresse et donnée doit être telle que montrée dans le Tableau 175.

Tableau 175 – Structure des éléments adresse et donnée

Nom de champ	Type	Description
Type ID	UINT	Type de l'élément encapsulé
Length	UINT	Longueur en octets des données à suivre
Data	Variable	Les données (si longueur >0)

Les ID de type d'adresse doivent être tels que montrés dans le Tableau 176.

Tableau 176 – ID de type d'adresse

ID de type d'adresse	Description
0x00	Null (utilisé pour les messages de l'UCMM)
0x01 à 0xA0	Réservés
0xA1	Basé sur la connexion (utilisé pour les messages connectés)
0xA2 à 0x8001	Réservés
0x8002	Type d'adresse "sequenced" (séquencée)
0x8003 à 0xFFFF	Réservés

Les ID de type de donnée doivent être tels que montrés dans le Tableau 177.

Tableau 177 – ID de type de donnée

ID de type de donnée	Description
0x00 à 0xB0	Réservés
0xB1	PDU de transport connectée
0xB2	Message non connecté
0xB3 à 0x7FFF	Réservés
0x8000	Sockaddr Info, émetteur vers cible
0x8001	Sockaddr Info, cible vers émetteur
0x8002 à 0xFFFF	Réservés

4.3.3.2 Type d'adresse

4.3.3.2.1 Null

Le type d'adresse "null" doit contenir seulement l'identificateur de type et la longueur, comme montré dans le Tableau 178. La longueur doit être zéro. Aucune donnée ne doit suivre la longueur. Comme le type d'adresse ne contient aucune information d'acheminement, il doit être utilisé lorsque le paquet de protocole lui-même contient toute information d'acheminement nécessaire. Le type "null" doit être utilisé pour les messages non connectés (Unconnected Messages).

Tableau 178 – Type d'adresse "null"

Nom de champ	Type	Valeur du champ
Type ID	UINT	0
Length	UINT	0

4.3.3.2.2 Connected

Ce type d'adresse doit être utilisé lorsque le protocole encapsulé est orienté connexion. Les données doivent contenir un identificateur de connexion, montré dans le Tableau 179.

NOTE Les identificateurs de connexion sont échangés dans le service Forward_Open service du Connection Manager.

Tableau 179 – Type d'adresse "connected"

Nom de champ	Type	Valeur du champ
Type ID	UINT	0xA1
Length	UINT	4
Data	UDINT	Identificateur de connexion

4.3.3.2.3 Type d'adresse "sequenced"

Ce type d'adresse doit être utilisé pour les données connectées de classe 0 et de classe 1. Les données doivent contenir un identificateur de connexion et un numéro de séquence, comme montré dans le Tableau 180. L'utilisation est décrite avec plus de détails en 11.3.3.

Tableau 180 – Type d'adresse "sequenced"

Nom de champ	Type	Valeur du champ
Type ID	UINT	0x8002
Length	UINT	8
Data	UDINT	Identificateur de connexion
	UDINT	Numéro de séquence

4.3.3.3 Types de données**4.3.3.3.1 Unconnected**

Le type de données qui encapsule un message non connecté doit être tel que montré dans le Tableau 181.

Tableau 181 – Type de données UCMM

Nom de champ	Type	Valeur du champ
Type ID	UINT	0xB2
Length	UINT	Longueur, en octets, du message non connecté
Data	Variable	Le message non connecté

Le format du message non connecté doit être le même que la PDU destinée au Message Router. L'en-tête UCMM, défini en 4.1.3, ne doit pas être présent. Le champ context dans l'en-tête d'encapsulation doit être utilisé pour la correspondance demande/réponse non connectée.

4.3.3.3.2 Connected

Le type de données qui encapsule une PDU de transport connectée doit être tel que montré dans le Tableau 182.

Tableau 182 – Type de données connected

Nom de champ	Type	Valeur du champ
Type ID	UINT	0xB1
Length	UINT	Longueur, en octets, de la PDU de transport
Data	Variable	La PDU de transport

4.3.3.3.3 Sockaddr info

Les éléments Sockaddr Info doivent être utilisés pour encapsuler des informations d'adresse de port de connexion nécessaires pour envoyer des datagrammes (les données connectées) entre la cible et l'émetteur. Ce sont des éléments séparés pour les informations de port de connexion émetteur vers cible et cible vers émetteur.

Les éléments Sockaddr Info doivent avoir la structure montrée dans le Tableau 183.

Tableau 183 – Éléments Sockaddr Info

Nom de champ	Type	Valeur du champ
Type ID	UINT	0x8000, 0x8001
Length	UINT	16 (octets)
Data	ARRAY	Voir ci-dessous.

La partie "données" du type Sockaddr Info doit suivre le format de la structure sockaddr_in définie dans la Specification Winsock, version 1.1. La structure de sockaddr_in est montrée ci-dessous:

```
struct sockaddr_in {
    short    sin_family;
    u_short  sin_port;
    struct   in_addr sin_addr;
    char     sin_zero[8];
};
```

Les champs à plusieurs octets dans la structure de sockaddr_in doivent être retournés dans l'ordre des octets TCP/IP, car il s'agit de l'ordre d'octets utilisé par Winsock API.

5 Syntaxe de transfert

5.1 Codage compact

5.1.1 Règles de codage

Le présent paragraphe décrit le moyen par lequel les types de données définis dans la présente norme doivent être codés/transférés sur le réseau de commande. La définition de syntaxe abstraite accompagnée d'un jeu particulier de règles de codage doit aboutir à la syntaxe de transfert. Pour les données d'utilisateur d'application, un seul jeu de règles de codage doit être défini («Compact Encoding» Codage compact), aboutissant à la syntaxe de transfert compacte.

Les règles de codage compact doivent commencer par les règles de codage définies dans l'ISO/CEI 8825-1. Le codage compact applique alors des règles d'optimisation, en commençant par l'unité de données de service (Service Data Unit, SDU) la plus extérieure et en progressant vers chaque SDU encapsulée successive. Le codage compact doit définir un mécanisme de codage plus efficace en réduisant la quantité des informations (surdébit) transférées entre les dispositifs.

La différence entre une valeur codée par codage compact et une valeur codée par codage ASN.1 doit être la suppression des champs décrivant le type et la longueur des informations. Les composants TAG et LENGTH d'une valeur codée par codage ASN.1 ne doivent pas être émis sur le réseau de commande. En outre, les règles du codage compact doivent indiquer que les règles d'ordre de classement des octets sont l'inverse de celles vues dans l'ASN.1.

Considérant les conditions énumérées en 5.1.2, des règles générales doivent être appliquées à une valeur codée par codage ASN.1 pour générer une valeur codée par codage compact. Les règles générales doivent être comme suit:

- supprimer les octets d'identificateur ("Identifier");
- supprimer les octets "TAG" spécifiés par ASN.1;
- supprimer les octets de longueur ("Length");
- supprimer les octets "LENGTH" spécifiés par ASN.1;
- inverser l'ordre des octets pour les octets de contenu multiples.

5.1.2 Contraintes de codage

La représentation d'une valeur variable utilisant le codage compact doit être possible avec les restrictions suivantes:

- le type de la variable doit être de longueur fixe et ne doit avoir aucun champ conditionnel ou facultatif;
- le codage d'une valeur donnée doit être représenté avec un nombre constant d'octets dérivés de la spécification de type de cette variable.

5.1.3 Exemples

5.1.3.1 Codage de BOOL

Le codage de BOOLEAN doit être accompli sur un seul OCTET comme décrit dans le Tableau 184.

Tableau 184 – Codage BOOLEAN

Si la valeur est:	Alors:
FALSE	le bit 0 de l'octet est 0 ('00'H)
TRUE	le bit 0 de l'octet est 1 ('01'H)

Un BOOL FALSE doit être représenté comme montré dans le Tableau 185.

Tableau 185 – Exemple de codage compact d'une valeur BOOL

Numéro d'octet	1 ^{er}
BOOL	00

5.1.3.2 Codage de SignedInteger

Le codage de SignedInteger doit être accompli comme décrit dans le Tableau 186

Tableau 186 – Codage des valeurs SignedInteger

Numéro d'octet	1 ^e	2 ^e	3 ^e	4 ^e	5 ^e	6 ^e	7 ^e	8 ^e
SINT	0LSB							
INT	0LSB	1LSB						
DINT	0LSB	1LSB	2LSB	3LSB				
LINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

NOTE L'exemple du Tableau 187 illustre le codage d'une variable de type DINT dont la valeur est 0x12345678.

Tableau 187 – Exemple de codage compact d'une valeur SignedInteger

Numéro d'octet	1 ^e	2 ^e	3 ^e	4 ^e
DINT	78	56	34	12

5.1.3.3 Encodage de UnsignedInteger

Le codage de UnsignedInteger doit être accompli comme décrit dans le Tableau 188.

Tableau 188 – Valeurs UnsignedInteger

Numéro d'octet	1 ^e	2 ^e	3 ^e	4 ^e	5 ^e	6 ^e	7 ^e	8 ^e
USINT	0LSB							
UINT	0LSB	1LSB						
UDINT	0LSB	1LSB	2LSB	3LSB				
ULINT	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

NOTE Le Tableau 189 illustre le codage d'une variable de type UDINT dont la valeur est 0xAABBCCDD.

Tableau 189 – Exemple de codage compact d'un UnsignedInteger

Numéro d'octet	1 ^e	2 ^e	3 ^e	4 ^e
UDINT	DD	CC	BB	AA

5.1.3.4 Codage de FixedLengthReal

Le codage de FixedLengthReal doit être accompli comme décrit dans le Tableau 190.

Tableau 190 – Valeurs FixedLengthReal

Numéro d'octet	1 ^e	2 ^e	3 ^e	4 ^e	5 ^e	6 ^e	7 ^e	8 ^e
REAL	0LSB	1LSB	2LSB	3LSB				
LREAL	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

Le Tableau 191 illustre le codage d'une variable (Float1) dont le type est REAL et dont la valeur est Float1: = 10,0.

NOTE 1 L'affectation de la valeur utilise la notation CEI 61131-3 notation. La valeur ASN.1 est {41200000'H} dans le format de l'IEEE: 1,25*23 ,l'exposant est 130 (biais 127), la fraction est 25.

Tableau 191 – Exemple de codage compact d'une valeur REAL

Contenus d'octet	0LSB	1LSB	2LSB	3LSB
REAL	00	00	20	41

Le Tableau 192 illustre le codage d'une variable (Float2) dont le type est LREAL et dont la valeur est Float2: = -100,0.

NOTE 2 La valeur ASN.1 est {C059000000000000'H} dans le format de l'IEEE: 1,562 5*26 ,l'exposant est 1 029 (biais 1 023), la fraction est 0,562 5.

Tableau 192 – Exemple de codage compact d'une valeur LREAL

Contenus d'octet	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB
LREAL	00	00	00	00	00	00	59	C0

5.1.3.5 Codages du temps (Time)

Le Tableau 193 illustre le codage de nombres réels (REAL) à longueurs fixes (fixedlengths).

Tableau 193 – Valeurs FixedLengthReal

Numéro d'octet	1 ^e	2 ^e	3 ^e	4 ^e	5 ^e	6 ^e	7 ^e	8 ^e
TIME	0LSB	1LSB	2LSB	3LSB				
DATE	0LSB	1LSB						
TIME_OF_DAY	0LSB	1LSB	2LSB	3LSB				
DATE_AND_TIME	0LSB Temps	1LSB Temps	2LSB Temps	3LSB Temps	0LSB Date	1LSB Date		
FTIME	0LSB	1LSB	2LSB	3LSB				
LTIME	0LSB	1LSB	2LSB	3LSB	4LSB	5LSB	6LSB	7LSB

5.1.3.6 Codages de chaîne (String)

Le présent paragraphe donne des exemples de codage compact appliqué à des valeurs de données STRING, STRING2, STRINGN et SHORT_STRING.

NOTE Le type de chaîne préférentiel pour les données de chaîne fournies par l'utilisateur est STRING2 en raison des exigences internationales relatives aux chaînes de caractères.

Le Tableau 194 donne une illustration générique du codage d'une valeur STRING.

Tableau 194 – Valeur STRING

	Contenu (charcount)		Contenu (contenu de chaîne)
	0LSB	1LSB	0LSB
STRING			

Caractères de 1 octet

Le Tableau 195 donne une illustration générique du codage d'une valeur STRING2.

Tableau 195 – Valeur STRING2

	Contenu (charcount)		Contenu (string2contents)	
	0LSB	1LSB	0LSB	1LSB
STRING2				

Caractères de 2 octets

Le Tableau 196 donne une illustration générique du codage d'une valeur STRINGN.

Tableau 196 – Valeur STRINGN

	Contenu (charsize)		Contenu (charcount)		Contenu (stringNcontents)	
	0LSB	1LSB	0LSB	1LSB	0LSB	NLSB
STRINGN						

Caractères de N octets

Le Tableau 197 donne une illustration générique du codage d'une valeur SHORT_STRING .

Tableau 197 – Valeur SHORT_STRING

	Contenu (charcount)	Contenu (short_string)
SHORT_STRING	0LSB	0LSB

Caractères de 1 octet

Le Tableau 198 illustre le codage d'une variable chaîne dont le contenu est "Mill". Des exemples de codage de tous les types de chaîne sont présentés. Le codage de caractère est spécifié dans l'ISO/CEI 10646. L'équivalent hexadécimal est: {‘4D696C6C’H} pour le codage 8 bits.

Le Tableau 198 code "Mill" comme un type STRING.

Tableau 198 – Exemple de codage compact d'une valeur STRING

	Contenu (charcount)		Contenu (contenu de chaîne)			
STRING	04	00	4D	69	6C	6C

Le Tableau 199 code "Mill" comme un type STRING2.

Tableau 199 – Exemple de codage compact d'une valeur STRING2

	Contenu (charcount)		Contenu (contenu string2)							
STRING2	04	00	4D	00	69	00	6C	00	6C	00

Le Tableau 200 code "Mill" comme un type SHORT_STRING.

Tableau 200 – Type SHORT STRING

	Contenu (charcount)		Contenu (contenu short_string)			
SHORT_STRING	04		4D	69	6C	6C

5.1.3.7 Codage de FixedLengthBitString

Le présent paragraphe donne des exemples de codage compact appliqué à des valeurs de données SWORD, WORD, DWORD, LWORD. La Figure 14 illustre les règles de placement des bits associées au codage compact d'un FixedLengthBitString.

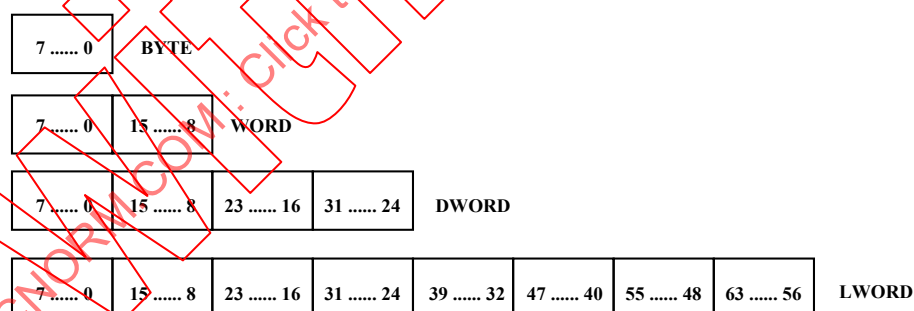


Figure 14 – Règles de placement de bits associées au codage compact de FixedLengthBitString

Les Figure 15 à Figure 18 illustrent le codage de SWORD, WORD, DWORD et LWORD.

Bits In Memory: 7 0
00001111

Compact Encoded BYTE
00001111 or 0x0F

Légende

Anglais	Français
Bits In Memory:	Bits en mémoire
Compact Encoded BYTE	BYTE à codage compact

Figure 15 – Exemple de codage compact d'un FixedLengthBitString SWORD

Bits In Memory: 15 0
00001111 11001111

Compact Encoded WORD
11001111 00001111 or 0xCF0F

Légende

Anglais	Français
Bits In Memory:	Bits en mémoire
Compact Encoded WORD	WORD à codage compact

Figure 16 – Exemple de codage compact d'un FixedLengthBitString WORD

Bits In Memory: 31 0
00001111 11001111 10110110 00111110

Compact Encoded DWORD
00111110 10110110 11001111 00001111 or 0x3EB6CF0F

Légende

Anglais	Français
Bits In Memory:	Bits en mémoire
Compact Encoded DWORD	DWORD à codage compact

Figure 17 – Exemple de codage compact d'un FixedLengthBitString DWORD

Bits In Memory:
 63 0
 11110000 11001111 10110110 00111110 11110000 00101101 00011110 00001111

Compact Encoded LWORD

00001111 00011110 00101101 11110000 00111110 10110110 11001111 11110000 or 0x0F1E2DF03EB6CFF0

Légende

Anglais	Français
Bits In Memory:	Bits en mémoire
Compact Encoded LWORD	LWORD à codage compact

Figure 18 – Exemple de codage compact d'un FixedLengthBitString LWORD

5.1.3.8 Codage de matrice (Array)

5.1.3.8.1 Codage de matrice unidimensionnelle

Le codage de Array doit utiliser les règles de codage pour les types de données pour chaque élément et concaténer les éléments qui composent la matrice. Les valeurs codées des éléments de la matrice doivent être codées dans le même ordre qu'elles sont déclarées dans la spécification de type ou de variable correspondante de l'ASN.1. Ces éléments peuvent être de n'importe quel type de données.

La définition au style ASN.1 d'une matrice unidimensionnelle dans le réseau de commande doit être:

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,
                        array_dimension_high_bound, NetworkData }
```

Assume the following array definition::

```
ARRAY1 ::= SEQUENCE OF {array_dimension_low_bound := 0,
                        array_dimension_high_bound := 1, UINT}
```

L'insertion des valeurs UINT {1,2} dans la définition de la matrice donne le codage spécifié dans le Tableau 201.

Tableau 201 – Exemple de codage compact d'un ARRAY à une seule dimension

Numéro d'octet	1 ^e	2 ^e	3 ^e	4 ^e
ARRAY	01	00	02	00

5.1.3.8.2 Codage de matrice bidimensionnelle

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,
                        array_dimension_high_bound,
                        SEQUENCE OF { array_dimension_low_bound,
                        array_dimension_high_bound, NetworkData } }
```

5.1.3.8.3 Codage de matrice tridimensionnelle

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound,
    array_dimension_high_bound,
    SEQUENCE OF { array_dimension_low_bound,
        array_dimension_high_bound,
        SEQUENCE OF { array_dimension_low_bound,
            array_dimension_high_bound, NetworkData } } }
```

Sachant que les données de réseau de commande peuvent comprendre des ElementaryData ou des DerivedData, une nouvelle spécification de type ou de variable peut être requise avant d'émettre les valeurs pour l'ARRAY.

Une matrice multidimensionnelle doit être vue sur la ligne comme une matrice à une seule dimension. L'ordre des éléments de la matrice doit être conservé par le biais d'une séquence de condensation/décompression, suivie des nœuds d'extrémité. La séquence suivie doit être pour accéder à la Nième dimension de la matrice pour toutes les valeurs des autres dimensions.

Cela doit être réalisé en accédant d'abord à la matrice avec toutes les dimensions mises à leurs valeurs d'indice initiales. Ensuite, la Nième dimension est incrémentée sur toutes ses valeurs d'indice. Lorsque la fin de la plage d'indices pour la Nième dimension est atteinte, la (N-1)ième dimension est incrémentée et la Nième dimension est mis à sa valeur d'indice initiale. Ce processus est répété jusqu'à ce que toutes les dimensions de la matrice aient atteint la fin de leurs plages d'indices. Ainsi, la matrice est condensée dans le tampon de messages comme une matrice à une seule dimension. Le même mode opératoire est suivi pour décompresser la matrice unidimensionnelle en une matrice multidimensionnelle

La matrice bidimensionnelle

```
ARRAY1 [0..1, 0..2] OF UINT := { { 1, 2, 3 },
    { 4, 5, 6 } }
```

conduit au train de données montré dans le Tableau 202 lorsqu'elle est condensée en une matrice à une seule dimension en suivant les règles de codage compact.

Tableau 202 – Exemple de codage compact d'un ARRAY à plusieurs dimensions

Numéro d'octet	1 ^e	2 ^e	3 ^e	4 ^e	5 ^e	6 ^e	7 ^e	8 ^e	9 ^e	10 ^e	11 ^e	12 ^e
ARRAY	01	00	02	00	03	00	04	00	05	00	06	00

5.1.3.9 Codage de Structure

Le codage d'une structure doit utiliser les règles de codage pour les types de données pour chaque élément et concatène les éléments qui composent la structure.

Les valeurs codées des éléments de la structure doivent être codées dans le même ordre qu'elles sont déclarées dans la spécification de type ou de variable correspondante de l'ASN.1. Ces éléments peuvent être de n'importe quel type de données.

```
STRUCT ::= SEQUENCE OF NetworkData -- May be different types
```

Sachant que NetworkData peut être constitué des ElementaryData ou des DerivedData, une nouvelle spécification de type ou de variable peut être requise avant d'émettre les valeurs pour le STRUCT.

Supposons la définition de structure suivante:

newStruct ::= SEQUENCE { BOOL,UINT,DINT }

L'insertion des valeurs {TRUE,'1234'H,'56789ABC'H} dans la structure donne le codage tel que spécifié dans le Tableau 203.

Tableau 203 – Exemple de codage compact d'une STRUCTURE

Numéro d'octet	1 ^e	2 ^e	3 ^e	4 ^e	5 ^e	6 ^e	7 ^e
newStruct	01	34	12	BC	9A	78	56

5.1.3.10 Exemples de formats de données codées complexes

5.1.3.10.1 Généralités

Les exemples en 5.1.3.10.2 et 5.1.3.10.3 montrent comment des formats de données plus complexes doivent être condensés. L'exemple 5.1.3.10.2 montre la condensation d'une matrice de structures. L'exemple en 5.1.3.10.3 montre comment une structure ayant un élément matrice est condensée.

5.1.3.10.2 Exemple 1: le codage d'une matrice de structures:

```
STRUCT1 ::= SEQUENCE OF {
    UINT      ele1;
    USINT     ele2;
    USINT     ele3;
    USINT     ele4;
    UINT      ele5
}

ARRAY1 [ 0..1 , 0..2 ] of STRUCT1 := {
    { { 1, 2, 3, 4, 5 }, { 6, 7, 8, 9, 10 },
      { 11, 12, 13, 14, 15 } },
    { { 15, 14, 13, 12, 11 }, { 10, 9, 8, 7, 6 },
      { 5, 4, 3, 2, 1 } } }
```

donne le train de données suivant:

```
[01][00][02][03][04][05][00] [06][00][07][08][09][0A][00]
[0B][00][0C][0D][0E][0F][00] [0F][00][0E][0D][0C][0B][00]
[0A][00][09][08][07][06][00] [05][00][04][03][02][01][00]
```

5.1.3.10.3 Exemple 2: le codage d'une structure ayant un élément matrice

```
STRUCT2 ::= SEQUENCE OF {
    UINT      ele1;
    ARRAY [ 0..2 ] of USINT array2;
    UINT      ele5;
}

STRUCT2 := { 1, { 2, 3, 4 }, 5 }
```

donne le train de données suivant:

```
[01][00] [02][03][04] [05][00]
```

5.2 Rapport relatif au type de données

5.2.1 Représentation de données d'objet

Des objets peuvent choisir de mettre en œuvre un mécanisme pour rendre compte du Data Type (type de données) d'un Attribut particulier ou pour émettre des informations relatives au

type accompagnées des données effectives. Le présent paragraphe définit le moyen par lequel les informations de typage des données sont acheminées.

La spécification des informations relatives au type de données utilise la méthodologie de l'ASN.1 spécifiée dans la série ISO/CEI 8824 et dans l'ISO/CEI 8825-1 avec les optimisations définies par le réseau de commande pour coder la production de **DataTypeSpecification** définie en 4.2.4.

L'optimisation définie par le réseau de commande est que l'octet Length d'un type NULL n'est pas codé.

NOTE Par exemple, le codage de 'abc [PRIVATE 1] IMPLICIT NULL' serait: 0xC1 (une étiquette sans octet de longueur).

5.2.2 Rapports relatifs aux types de données élémentaires

Les types de données élémentaires doivent être identifiés en utilisant les codes d'identification définis dans le Tableau 204. Ces codes doivent définir le codage des membres de primitive de la production de **DataTypeSpecification**. Le réseau de commande spécifie que les types NULL de l'ASN.1 ne doit pas rapporter l'octet de longueur de zéro (0).

Tableau 204 – Codes d'identification et descriptions de types de données élémentaires

Nom de Data type	Code de Data type (en hexadécimal)	Description de Data type
BOOL	C1	Booléen logique avec les valeurs TRUE et FALSE
SINT	C2	Valeur entière signée de 8 bits
INT	C3	Valeur entière signée de 16 bits
DINT	C4	Valeur entière signée de 32 bits
LINT	C5	Valeur entière signée de 64 bits
USINT	C6	Valeur entière non signée de 8 bits
UINT	C7	Valeur entière non signée de 16 bits
UDINT	C8	Valeur entière non signée de 32 bits
ULINT	C9	Valeur entière non signée de 64 bits
REAL	CA	Valeur en virgule flottante de 32 bits
LREAL	CB	Valeur en virgule flottante de 64 bits
STIME	CC	Information de temps synchrone
DATE	CD	Information de date
TIME_OF_DAY	CE	Heure du jour
DATE_AND_TIME	CF	Date et heure du jour
STRING	D0	chaîne de caractères (1 octet par caractère)
SWORD	D1	chaîne binaire 8 bits
WORD	D2	chaîne binaire 16 bits
DWORD	D3	chaîne binaire 32 bits
LWORD	D4	chaîne binaire 64 bits
STRING2	D5	chaîne de caractères (2 octets par caractère)
FTIME	D6	Durée (haute résolution)

Nom de Data type	Code de Data type (en hexadécimal)	Description de Data type
LTIME	D7	Durée (longue)
ITIME	D8	Durée (courte)
STRINGN	D9	chaîne de caractères (N octets par caractère)
SHORT_STRING	DA	chaîne de caractères (1 octet par caractère, indicateur de longueur 1 octet)
TIME	DB	Durée (millisecondes)
EPATH	DC	Segments de chemin
STRINGI	DE	Chaîne de caractères internationaux

5.2.3 Rapports relatifs aux types de données construits

5.2.3.1 Définition du type Structure

5.2.3.1.1 Généralités

Le réseau de commande définit deux méthodes différentes pour rapporter les définitions du type Structure:

- Codage formel (FormalStructTypeSpec);
- Codage abrégé (AbbreviatedStructTypeSpec).

Le codage formel doit être utilisé pour fournir un rapport détaillé de la définition complète de la structure, y compris la définition complète de tous les types de données des composants. Le codage abrégé doit être utilisé pour spécifier une forme *plus courte* de la définition de la structure. La forme *plus courte* ne doit pas inclure les types de données associés aux composants de la structure.

5.2.3.1.2 Codage formel pour les informations relatives au type de structure

Les exemples en 5.2.3.1.3 et en 5.2.3.1.4 illustrent le codage formel pour les spécifications des types de structure. Il s'agit en fait d'un exemple du codage de la production de FormalStructTypeSpec définie en 4.2.4.

5.2.3.1.3 Exemple 1

Le Tableau 205 illustre le codage de la définition de structure suivante.

```
STRUCT ::= SEQUENCE OF { BOOL, UINT, DINT }
```

Tableau 205 – Exemple 1 de codage formel d'une spécification de type structure

Type STRUCT	Longueur du type	Types de composant	Types de composant	Types de composant
		BOOL	UINT	DINT
A2	03	C1	C7	C4
NOTE Les types NULL IMPLICIT issus de la production de DataTypeSpecification ne sont pas suivis d'un octet de longueur de zéro (0).				

5.2.3.1.4 Exemple 2

La Figure 19 illustre le codage de la définition de structure suivante.

```
STRUCT_MAIN ::= SEQUENCE OF { UINT, STRUCT_SUB, INT }
```

with subelement STRUCT_SUB defined as:

```
STRUCT_SUB ::= SEQUENCE OF { UINT, SINT, INT }
```

Struct Type	Type Length	Component Types						
		UINT	Struct Type	Type Length	Nested Structure Component Types			
A2	07	C7	A2	03	UINT	SINT	INT	INT
		C7			C2	C3	C3	C3

Légende

Anglais	Français
Struct Type	Type structure
Type Length	Longueur du type
Component Types	Type de composant
Nested Structure Component Types	Type de composant de structure imbriquée

Figure 19 – Exemple 2 de codage formel d'une spécification de type structure

5.2.3.1.5 Codage abrégé pour les informations relatives au type de structure

L'exemple en 5.2.3.1.6 illustre le codage abrégé pour les spécifications du type de structure. Il s'agit en fait d'un exemple du codage de la production de `AbbreviatedStrucTypeSpec` définie en 4.2.4.

L'UINT défini au sein de la production de `AbbreviatedStrucTypeSpec` doit être initialisé avec une valeur de contrôle de redondance cyclique (CRC) de 16 bits (voir CEI 61158-4-2:2010). Ceci peut être utilisé par la logique d'application pour déterminer si, oui ou non, le format de la structure a changé. Le train d'octets utilisé pour produire le CRC est la spécification de type structure formellement codée (`FormalStrucTypeSpec`) réelle.

5.2.3.1.6 Exemple

La Figure 20 montre le codage abrégé de la définition de structure présentée en 5.2.3.1.5:

Struct Type	Type Length	UINT containing CRC	
A0	02	C7	26

Légende

Anglais	Français
Struct Type	Type structure
Type Length	Longueur du type
UINT containing CRC	UINT contenant CRC

Figure 20 – Exemple de codage abrégé d'une spécification de type structure

NOTE Les algorithmes présentés dans la présente NORME sont utilisés pour générer la valeur CRC de 0x26C7 issue de la spécification du type Formally Encoded (formellement codé): [A2][07][C7][A2][03][C7][C2][C3][C3].

5.2.3.2 Définition du type Array

Deux méthodes différentes pour rapporter les définitions du type Array (matrice) doivent être:

- Codage formel (FormalArrayTypeSpec);
- Codage abrégé (AbbreviatedArrayTypeSpec).

Le codage formel doit être utilisé pour fournir un rapport détaillé de la définition complète de la matrice, y compris le contenu de données et les dimensions de la matrice. Le codage abrégé doit être utilisé pour spécifier une forme *plus courte* de la définition de la matrice. Cette forme *plus courte* ne doit pas inclure les informations spécifiant les dimensions de la matrice.

5.2.3.3 Codage formel pour les informations relatives au type de matrice**5.2.3.3.1 Notes communes**

NOTE 1 Le présent paragraphe ne contient pas d'exigences normatives.

NOTE 2 Les exemples en 5.2.3.3.2 et en 5.2.3.3.3 illustrent le codage formel pour les spécifications des types de structure. Il s'agit en fait d'un exemple du codage de la production de FormalArrayTypeSpec définie en 4.2.4.

5.2.3.3.2 Exemple 1

La Figure 21 montre le codage formel de la définition de matrice suivante:

```
ARRAY1 ::= SEQUENCE OF { array_dimension_low_bound:= 0,
                          array_dimension_high_bound:= 9,
                          UINT
```

Array Type	Type Length	Lower Bound Tag	Lower Bound Length	Lower Bound	Upper Bound Tag	Upper Bound Length	Upper Bound	UINT
A3	07	80	01	00	81	01	09	C7

Légende

Anglais	Français
Array Type	Type array (matrice)
Type Length	Longueur de type
Lower Bound Tag	Étiquette de borne inférieure
Lower Bound Length	Longueur de borne inférieure
Lower Bound	Borne inférieure
Upper Bound Tag	Étiquette de borne supérieure
Upper Bound Length	Longueur de borne supérieure
Upper Bound	Borne supérieure
UINT	UINT

Figure 21 – Exemple 1 de codage formel d'une spécification de type array

NOTE Les types NULL IMPLICIT issus de la production de DataTypeSpecification ne sont pas suivis d'un octet de longueur de zéro (0).

5.2.3.3.3 Exemple 2

La Figure 22 montre le codage de la définition de matrice suivante:

```

ARRAY1 ::= SEQUENCE OF { array_dimension_low_bound:= 0,
                           array_dimension_high_bound:= 19,
                           SEQUENCE OF { array_dimension_low_bound:= 0,
                                           array_dimension_high_bound:= 255,
                                           STRUCT_ELE } }
STRUCT_ELE } }

```

dans laquelle STRUCT_ELE est défini comme étant:

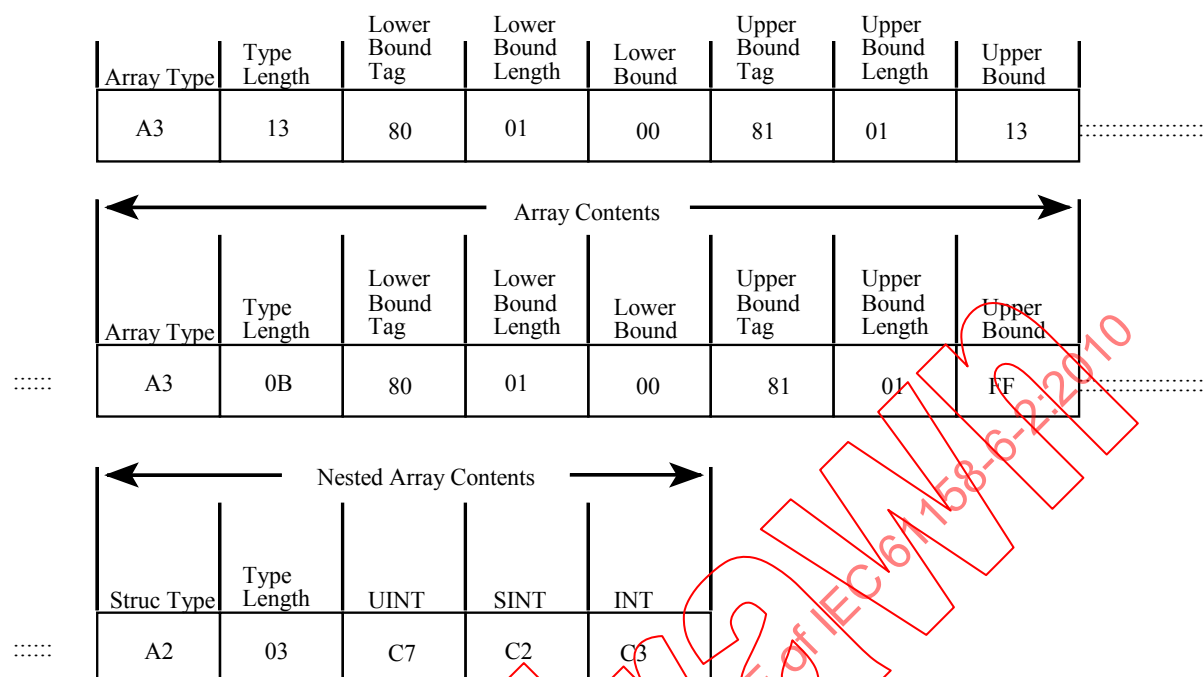
```

STRUCT_ELE ::= SEQUENCE OF { UINT, SINT, INT }

```

Formal Encoding: [A3][13][80][01][00][81][01][13][A3][0B][80][01][00][81][01][FF][A2][03][C7][C2][C3]

Description:



Légende

Anglais	Français
Formal encoding	Codage formel
Array Type	Type array (matrice)
Type Length	Longueur de type
Lower Bound Tag	Étiquette de borne inférieure
Lower Bound Length	Longueur de borne inférieure
Lower Bound	Borne inférieure
Upper Bound Tag	Étiquette de borne supérieure
Upper Bound Length	Longueur de borne supérieure
Upper Bound	Borne supérieure
Array Contents	Contenu de la matrice
Array Type	Type array (matrice)
Type Length	Longueur de type
Lower Bound Tag	Étiquette de borne inférieure
Lower Bound Length	Longueur de borne inférieure
Lower Bound	Borne inférieure
Upper Bound Tag	Étiquette de borne supérieure
Upper Bound Length	Longueur de borne supérieure
Upper Bound	Borne supérieure
Nested Array Contents	Contenu imbriqué de la matrice
Struc Type	Type structure
Type Length	Longueur de type

Figure 22 – Exemple 2 de codage formel d'une spécification de type array

NOTE Les types NULL IMPLICIT issus de la production de DataTypeSpecification ne sont pas suivis d'un octet de longueur de zéro (0).

5.2.3.4 Codage d'étiquette abrégé pour les informations relatives au type array

5.2.3.4.1 Notes communes

NOTE 1 Le présent paragraphe ne contient pas de références normatives.

NOTE 2 Le codage abrégé d'un type Array ne doit pas inclure les informations spécifiant les dimensions de la matrice. Il s'agit en fait d'un exemple du codage de la production de `AbbreviatedArrayTypeSpec` définie en 4.2.4.

5.2.3.4.2 Exemple 1

La Figure 23 montre le codage abrégé de la définition de matrice suivante:

```
ARRAY2 ::= SEQUENCE OF { array_dimension_low_bound:= 0,
                          array_dimension_high_bound:= 9,
                          UINT }
```

Array Type	Type Length	UINT
A1	01	C7

Légende

Anglais	Français
Struct Type	Type structure
Type Length	Longueur du type
UINT	UINT

Figure 23 – Exemple 1 de codage abrégé d'une spécification de type array

NOTE Les types `NULL IMPLICIT` issus de la production de `DataTypeSpecification` ne sont pas suivis d'un octet de longueur de zéro (0).

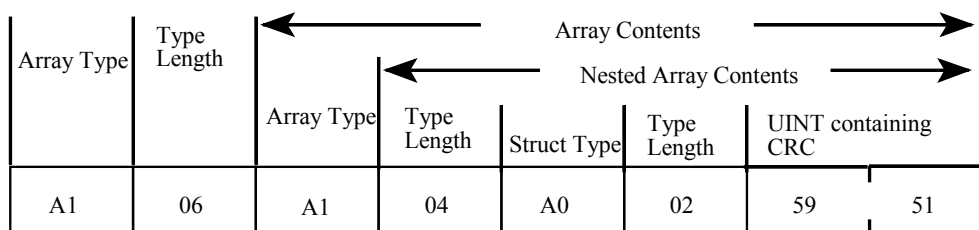
5.2.3.4.3 Exemple 2

La Figure 24 montre le codage abrégé de la définition de matrice suivante:

```
ARRAY ::= SEQUENCE OF { array_dimension_low_bound:= 0,
                        array_dimension_high_bound:= 19,
                        SEQUENCE OF { array_dimension_low_bound:= 0,
                        array_dimension_high_bound:= 899,
                        STRUCT_ELEMENT } }
```

dans laquelle `STRUCT_ELEMENT` est défini comme étant:

```
STRUCT_ELEMENT ::= SEQUENCE OF { UINT, SINT, INT }
```



Légende

Anglais	Français
Array Type	Type array (matrice)
Type Length	Longueur de type
Array Contents	Contenu de la matrice
Nested Array Contents	Contenu imbriqué de la matrice
Struc Type	Type structure
UINT containing CRC	UINT contenant CRC

Figure 24 – Exemple 2 de codage abrégé d'une spécification de type array

6 Structure des diagrammes d'états de protocole de FAL

Les diagrammes de protocole et de services d'interfaces à la FAL sont spécifiés dans le présent paragraphe. Les conventions utilisées pour les descriptions sont données dans la partie générique de la présente Norme CEI.

Ce bus de terrain suit la structure indiquée pour le bus de terrain de Type 1 avec les caractéristiques spécifiques suivantes:

- Il n'y a pas de définition formelle de Diagramme de contexte AP (AP-Context Machine)
- Il y a un diagramme FSPM formellement défini qui sert d'interface entre l'utilisateur de la FAL et le diagramme ARPM.
- Il y a des diagrammes ARPM de deux types différents:
 - 1) un diagramme ARPM pour les relations d'application sans connexion
 - 2) sept diagrammes ARPM pour les relations d'application orientées connexion
- Le diagramme DMPM est défini à l'interface de la couche liaison de données de Type 2.

7 Diagramme d'état de contexte AP

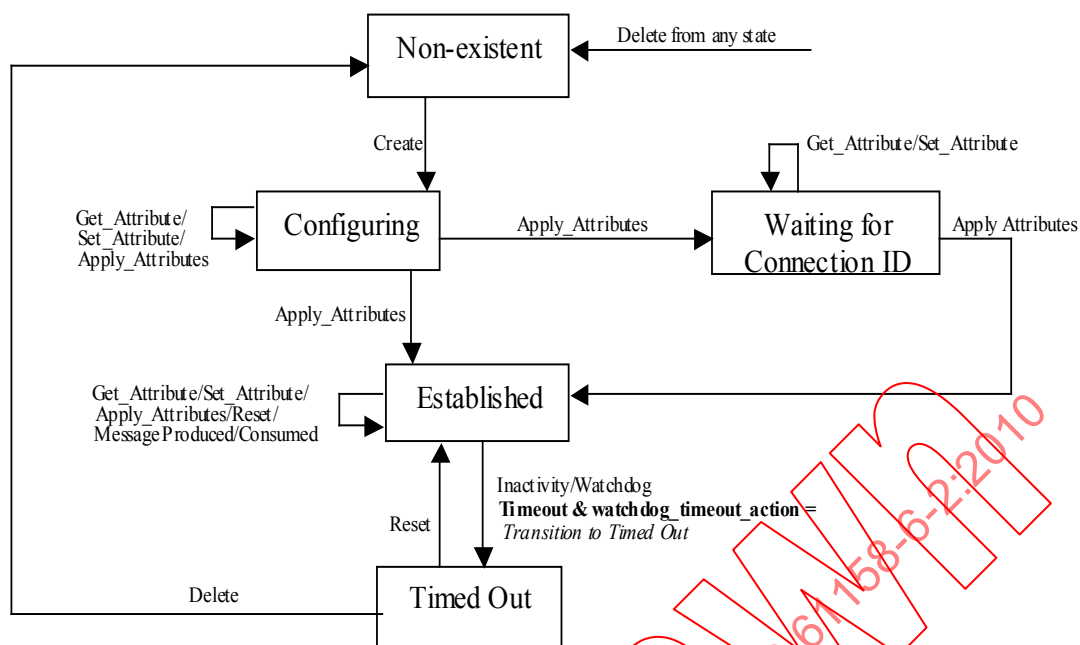
7.1 Vue d'ensemble

Le Type 2 prend en charge le diagramme d'états de contexte AP y spécifié, lorsque l'objet Connection est également pris en charge.

7.2 Diagramme d'état d'objet Connection

7.2.1 Comportement d'instance "I/O Connection"(Connexion E/S)

La Figure 25 donne une vue d'ensemble générale du comportement associé à un objet **I/O Connection** (attribut **instance_type** = E/S).



Légende

Anglais	Français
Delete from any state	Supprimer de n'importe quel état
Non-existent	Inexistant
Create	Créer
Delete	Supprimer
Get_Attribute/Set_Attribute	«Récupérer attribut/Établir attribut»
Get_Attribute/ Set_Attribute/ Apply_Attributes	«Récupérer attribut/Établir attribut/Appliquer attributs»
Configuring	Configuring
Apply_Attributes	Appliquer attributs
Waiting for Connection ID	En attente d'un identificateur de connexion
Established	Établie
Get_Attribute/Set_Attribute/ Apply_Attributes/Reset/	«Récupérer attribut/Établir attribut/Appliquer attributs/Réinitialiser/»
Message Produced/Consumed	Message produit/consommé
Apply_Attributes	Appliquer attributs
Timed Out	Temporisée
Inactivity/Watchdog Timeout & watchdog_timeout_action = Transition to Timed Out	Temporisation Inactivité/Chien de garde ET action de temporisation de chien de garde = Passage à « temporisée »
Reset	Réinitialiser

Figure 25 – Diagramme de transition d'états d'objet Connection "I/O"

Important: Le Tableau 206 fournit une State Event Matrix (Matrice d'événements d'états) pour un objet I/O Connection et il convient de baser les mises en œuvre sur ces informations. Cette State Event Matrix n'impose pas de règles concernant la logique interne spécifique à un produit. Une tentative d'accéder à la classe Connection ou à une instance d'objet Connection

peut nécessiter de passer par une vérification spécifique à un produit. Il peut en résulter un scénario d'erreur qui n'est pas indiqué par la SEM dans le Tableau 206. Il peut aussi en résulter des indications supplémentaires spécifiques à un produit fournies par un objet Connection à une application et/ou à un objet d'application spécifique. Le point à garder à l'esprit est que l'objet Connection doit manifester le comportement visible de l'extérieur qui est spécifié par la SEM et les définitions d'attributs.

Tableau 206 – Matrice d'événements d'états d'objet "I/O Connection"

Événement	État d'objet "I/O Connection"				
	Non-existant (inexistant)	Configuring (en configuration)	Waiting For Connection ID (en attente de l'ID de connexion)	Established (établie)	Timed Out (temporisée)
La classe Connection reçoit une demande Create.	La classe instancie un objet Connection. Mettre instance_type à I/O. Mettre tous les autres attributs aux valeurs par défaut. Passer à Configuring	Non applicable	Non applicable	Non applicable	Non applicable
La classe Connection reçoit une demande Delete.	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Libérer toutes les ressources associées. Passer à Non-existant	Libérer toutes les ressources associées. Passer à Non-existant	Libérer toutes les ressources associées. Passer à Non-existant	Libérer toutes les ressources associées. Passer à Non-existant
Set_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse	Si demande de modifier produced_connection_id ou consumed_connection_id, valider la valeur et gérer la demande. Retourner la réponse appropriée. S'il s'agit d'une demande d'accéder à un attribut autre que le produced_connection_id ou le consumed_connection_id, retourner une réponse d'erreur (Error Response) dont le Code d'erreur général est mis à 0C _{hex} . (L'objet ne peut pas accomplir le service demandé dans son mode/état actuel.)	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse appropriée.
Get_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse

Événement	État d'objet "I/O Connection"				
	Non-existent (inexistant)	Configuring (en configuration)	Waiting For Connection ID (en attente de l'ID de connexion)	Established (établie)	Timed Out (temporisée)
Réinitialisation	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général =)	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0C _{hex})	Annuler le temporisateur Inactivity/Watchdog Timer courant. En utilisant la valeur de l'attribut expected_packet_rate, redémarrer le temporisateur Inactivity/Watchdog Timer. Une réponse de succès est retournée même si un Inactivity/Watchdog Timer n'est pas utilisé par l'objet Connection (Classe de transport Client 0, expected_packet_rate = 00C _{hex}).	En utilisant la valeur de l'attribut expected_packet_rate, démarrer le temporisateur Inactivity/Watchdog Timer <u>et revenir à l'état Established</u> . Si l'attribut expected_packet_rate a été mis à zéro (0) alors que la connexion était dans l'état Timed Out (temporisée), revenir juste à l'état Established sans activer un temporisateur Inactivity/Watchdog Timer.

Événement	État d'objet "I/O Connection"				
	Non-existent (inexistant)	Configuring (en configuration)	Waiting For Connection ID (en attente de l'ID de connexion)	Established (établie)	Timed Out (temporisée)
Apply_ Attributes	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Délivrer l'objet Connection à l'application qui valide les informations d'attribut. Si l'un ou l'autre des attributs connection_id (produit ou consommé) nécessite d'être configuré et ne peut pas être généré par ce module, exécuter toutes les autres étapes nécessaires pour configurer la connexion, retourner une réponse de réussite (Successful) et passer à l'état Waiting For Connection ID. L'incapacité de générer une valeur d'ID de connexion produite ou consommée N'EST PAS rapportée comme une erreur. Si tous les attributs sont configurés en toute validité, exécuter toutes les étapes nécessaires pour satisfaire à cette connexion, démarrer tous les temporisateurs requis et passer à l'état Established. Si une erreur est détectée, une réponse d'erreur est retournée et la connexion reste dans l'état Configuring et, si une connexion client avec une valeur de déclencheur de production de 0 ou 1 ("Cyclic" ou "Change of State"), produire les données initiales.	Si l'un ou l'autre des attributs connection_id (produit ou consommé) nécessite encore d'être configuré et ne peut pas être généré par ce module, retourner une réponse de réussite (Successful) et rester dans l'état Waiting For Connection ID. L'incapacité de générer une valeur d'ID de connexion produite ou consommée N'EST PAS rapportée comme une erreur. Si l'attribut produced_connection_id et/ou l'attribut consumed_connection_id est/sont configuré(s) en toute validité, passer à l'état Established et retourner une réponse de succès (Successful) et, si une connexion client avec une valeur de déclencheur de production de 0 ou 1 (Cyclic or Change of State), produire les données initiales.	Toutes les modifications ont lieu immédiatement dès que la connexion est passée à l'état Established. Retourner l'erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0C _{hex})	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0C _{hex})

Événement	État d'objet "I/O Connection"				
	Non-existent (inexistant)	Configuring (en configuration)	Waiting For Connection ID (en attente de l'ID de connexion)	Established (établie)	Timed Out (temporisée)
Receive_Data	Non applicable	Rejeter le message	Rejeter le message	Si un message, complet valide ² a été reçu, réinitialiser le temporisateur Inactivity/Watchdog Timer ³ et délivrer le message I/O à l'application. <u>Un objet Connection doit manifester le comportement visible de l'extérieur qui est associé à l'état actuel de ses attributs (voir Règles d'accès).</u> S'il s'agit d'une partie fragmentée d'un message E/S, effectuer le traitement comme spécifié par le sous-réseau.	Rejeter le message.
Send_Message	Non applicable	Retourner erreur interne ne pas envoyer le message.	Retourner erreur interne ne pas envoyer le message.	Émettre le message E/S complet ou le fragment comme requis par le sous-réseau. S'il s'agit d'une connexion client et l'attribut expected_packet_rate est différent de zéro, redémarrer le temporisateur Transmission Trigger Timer.	Retourner erreur interne ne pas envoyer le message.
Inactivity/ Watchdog Timer expire	Non applicable	Non applicable	Non applicable	Examiner l'attribut watchdog_timeout_action de la connexion et accomplir l'action indiquée. Si l'attribut watchdog_timeout_action indique que la connexion doit rester dans l'état Established (Auto Reset), redémarrer immédiatement le temporisateur Inactivity/Watchdog Timer.	Non applicable

- a Si la configuration indique qu'un ID de message a besoin d'être alloué et il n'existe pas d'ID de message disponible dans le Groupe de message spécifié, il est retourné une réponse d'erreur dont le Code d'erreur général indique *Resource Unavailable* (02_{hex}). Si une valeur d'attribut d'objet Connection a passé avec succès la vérification de plage lorsqu'elle a été configurée initialement, mais la valeur d'attribut est en conflit avec un autre élément d'information dans le nœud lorsque la demande Apply est traitée, il est retourné une réponse d'erreur dont le Code d'erreur général est mis à *Valeur d'attribut non valide* (09_{hex}) et dont le code supplémentaire (Additional Code) est mis à l'ID d'attribut de l'objet Connection incriminé.
- b L'objet Connection vérifie que la longueur du message E/S reçu est inférieure ou égale à l'attribut `consumed_connection_size` avant de traiter le message. Si la longueur du message reçu est inférieure ou égale à l'attribut `consumed_connection_size`, l'objet I/O Connection réinitialise le temporisateur Inactivity/Watchdog, manifeste le comportement visible de l'extérieur indiqué par ses réglages d'attribut et délivre le message à l'Application. Si la longueur du message reçu est supérieure à l'attribut `consumed_connection_size`, l'objet I/O Connection doit immédiatement rejeter le message et interrompre tout traitement ultérieur. Il s'agit de la seule validation de contenu de message accomplie par l'objet I/O Connection. La validation ultérieure doit être accomplie par l'Application.
- c Si un message fragmenté est en train d'être reçu, le temporisateur Inactivity/Watchdog Timer n'est pas réinitialisé tant que le message tout entier n'a pas été valablement reçu.

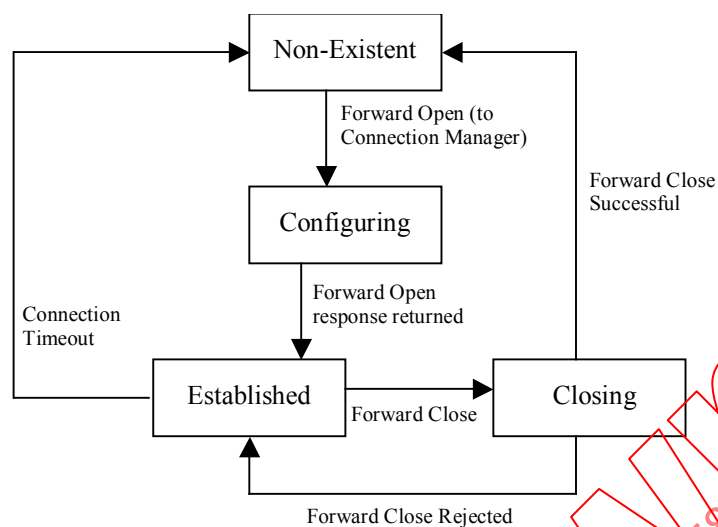
NOTE Les Règles d'accès d'attribut sont spécifiées dans la CEI 61158-5-2:2010, paragraphe 6.2.3.2.1.4

Important: L'événement `Receive_Data` n'est délivré à une I/O Connection que lorsqu'il est reçu un message dont le champ Connection Identifier concorde avec l'attribut `consumed_connection_id`. S'il est reçu un message dont le champ Connection Identifier ne concorde avec aucun attribut `consumed_connection_id` d'un objet Connection Established, le message est rejeté. Les identificateurs de connexion (Connection Identifier) sont spécifiques à un type de sous-réseau.

Si une mise en œuvre détecte qu'elle ne prend pas en charge un service de messagerie explicite indiqué dans le Tableau 206, il est retourné une réponse d'erreur spécifiant *Service Not Supported* (Code d'erreur général 08).

7.2.2 Comportement d'instance de "Bridged Connection"

La Figure 26 fournit une vue d'ensemble générale du comportement associé à un objet "Bridged Connection" (attribut `instance_type` = Bridged). Les connexions pontées servent à effectuer des connexions *offlink* (hors liaison). La messagerie tant E/S qu'explicite peut être accomplie avec l'aide de ce type de connexion. La définition de l'objet Connection Manager donne plus de détails sur ces types de connexions.



Légende

Anglais	Français
Non-Existent	Inexistant
Configuring	En configuration
Established	Établie
Connection Timeout	Temporisation de connexion Timeout
Forward Open (to Connection Manager)	Forward Open (Transmettre « ouvrir ») (à l'attention du gestionnaire de connexion)
Forward Open response returned	Réponse à Forward Open retournée
Closing	En fermeture
Forward Close	Forward Close (Transmettre « fermer »)
Forward Close Rejected	Le service «Transmettre « fermer » est rejetée
Forward Close Successful	Transmettre « fermer » a abouti.

Figure 26 –Diagramme de transition d'états d'un objet Connection "Bridged"

Le Tableau 207 fournit une State Event Matrix détaillée pour un objet Connection "Bridged ".

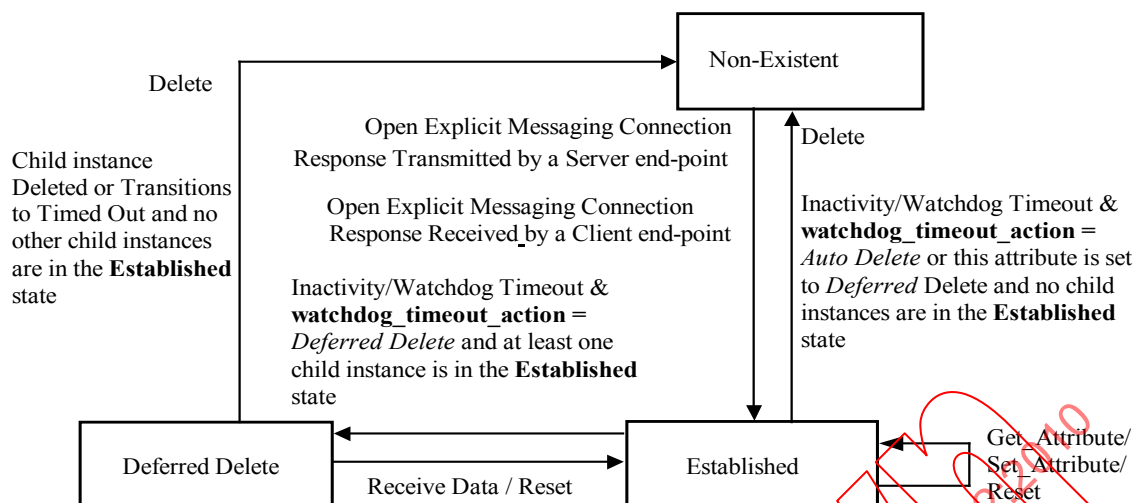
Tableau 207 – Matrice d'événements d'états d'une connexion "Bridged "

Événement	État de l'objet Connection "Bridged "			
	Non-existent (inexistant)	Configuring (en configuration)	Established (établie)	Closing (en cours de fermeture)
Le Connection Manager reçoit une demande Forward_Open.	La classe Connection instancie un objet Connection. Mettre instance_type à Bridged . Mettre les attributs à la valeur délivrée par le service Forward_Open ou à la valeur par défaut pour les connexions pontées. Passer à Configuring .	Non applicable	Non applicable	Non applicable
Le Connection Manager reçoit la notification que l'établissement de connexion à la cible est achevé .	Non applicable	Passer à Established	Non applicable	Non applicable
Le Connection Manager reçoit une demande Forward_Close.	Non applicable	Ignorer l'événement.	Passer à Closing	Ignorer l'événement.
Le Connection Manager reçoit une réponse Forward_Close.	Non applicable	Libérer toutes les ressources et passer à Non-Existent .	Libérer toutes les ressources et passer à Non-Existent .	Libérer toutes les ressources et passer à Non-Existent .
Delete (c'est-à-dire effacer, détruire)	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Libérer toutes les ressources et passer à Non-Existent .	Libérer toutes les ressources et passer à Non-Existent .	Libérer toutes les ressources et passer à Non-Existent .
Get_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse appropriée.
Set_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})			
Reset (réinitialisation)	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Erreur: Service non pris en charge (Code d'erreur général 08 _{hex})	Erreur: Service non pris en charge (Code d'erreur général 08 _{hex})	Erreur: Service non pris en charge (Code d'erreur général 08 _{hex})
Apply_Attributes	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Erreur: Service non pris en charge (Code d'erreur général 08 _{hex})	Erreur: Service non pris en charge (Code d'erreur général 08 _{hex})	Erreur: Service non pris en charge (Code d'erreur général 08 _{hex})

Événement	État de l'objet Connection "Bridged "			
	Non-existent (inexistant)	Configuring (en configuration)	Established (établie)	Closing (en cours de fermeture)
Receive_Data	Non applicable	Ignorer l'événement.	Invoquer le service envoi de l'objet Connection sur le port de destination, en transmettant les données reçues.	Ignorer l'événement.
Send_Message	Non applicable	Ignorer l'événement.	Envoyer les données sur le sous-réseau.	Ignorer l'événement.
Le temporisateur Inactivity/Watchdog Timer expire.	Non applicable	Non applicable	Libérer toutes les ressources et passer à Non- Existent.	Libérer toutes les ressources et passer à Non- Existent.
NOTE Les Règles d'accès d'attribut sont spécifiées dans la CEI 61158-5-2:2010, paragraphe 6.2.3.2.1.4				

7.2.3 Comportement de l'instance de connexion de messagerie explicite

La Figure 27 donne une vue d'ensemble générale du comportement associé à un objet **Connection "Explicit Messaging"** (attribut **instance_type** = Explicit Messaging).



Légende

Anglais	Français
Non-Existent	Inexistant
Established	Établie
Inactivity/Watchdog Timeout & watchdog_timeout_action = Auto Delete or this attribute is set to Deferred Delete and no child instances are in the Established state	Temporisation inactivité/chien de garde ET action de temporisation chien de garde = autodestruction ou cet attribut est mis à Suppression Différé et aucune instance enfant ne se trouve à l'état « Établie »
Get_Attribute/ Set_Attribute/ Reset	Récupérer attribut/établir attribut/Réinitialiser
Open Explicit Messaging Connection Response Transmitted by a Server end-point	Un point d'extrémité serveur a émis une réponse à une demande d'ouvrir connexion de messagerie explicite
Open Explicit Messaging Connection Response Received by a Client end-point	Un point d'extrémité client a reçu une réponse à une demande d'ouvrir connexion de messagerie explicite
Delete	Supprimer
Deferred Delete	Suppression différée
Inactivity/Watchdog Timeout & watchdog_timeout_action = Deferred Delete and at least one child instance is in the Established state	Temporisation inactivité/chien de garde ET action de temporisation chien de garde = Suppression différée ET au moins une instance enfant se trouve à l'état « Établie »
Child instance Deleted or Transitions to Timed Out and no other child instances are in the Established state	L'instance enfant est supprimée ou passe à « temporisée » et aucune autre instance enfant ne se trouve à l'état "éta4blie"
Receive Data / Reset	Recevoir des données/Réinitialiser

Figure 27 –Diagramme de transitions d'états d'un objet Connection "Explicit Messaging"

Le Tableau 208 fournit une State Event Matrix détaillée pour un objet Connection "Explicit Messaging". Il convient de bases les mises en œuvre sur les informations du Tableau 208.

Tableau 208 – Matrice d'événements d'états d'une connexion "Explicit Messaging"

Événement	État de l'objet Connection "Explicit Messaging"		
	Non-existent (inexistant)	Established (établie)	Deferred Delete (suppression différée).
L'UCMM reçoit une demande/réponse d'ouvrir une connexion de messagerie explicite (Open Explicit Messaging Connection) et invoque le service Create de la classe de connexion	Si possible, la classe instancie l'objet Connection. Mettre l'attribut Set instance_type à <i>Explicit Messaging^a</i> . Les autres attributs sont automatiquement configurés à l'aide des informations contenues dans la demande/réponse de "Open Explicit Messaging Connection". Passer à Established . Si demande reçue, émettre la réponse "Open Explicit Messaging Connection".	Non applicable	Non applicable
L'UCMM reçoit une demande Close et invoque le service Delete de la classe de connexion.	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Libérer toutes les ressources associées. Passer à Non-existent .	Libérer toutes les ressources associées. Passer à Nonexistent .
La classe Connection reçoit une demande Delete.	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Libérer toutes les ressources associées. Passer à Non-existent .	Libérer toutes les ressources associées. Passer à Nonexistent .
Set_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse appropriée.
Get_Attribute_Single	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse appropriée.	Valider/gérer la demande en fonction de la logique interne et selon les Règles d'accès («Access Rules») Retourner la réponse appropriée.
Reset (réinitialisation)	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Annuler le temporisateur Inactivity/Watchdog Timer courant. En utilisant la valeur de l'attribut expected_packet_rate , redémarrer le temporisateur Inactivity/Watchdog Timer.	En utilisant la valeur de l'attribut expected_packet_rate , redémarrer le temporisateur Inactivity/Watchdog Timer et revenir à l'état Established .
Apply_Attributes	Erreur: L'objet n'existe pas (Code d'erreur général 16 _{hex})	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0C _{hex})	Erreur: L'objet ne peut pas accomplir le service demandé dans son mode/état actuel. (Valeur du Code d'erreur général = 0C _{hex})

Événement	État de l'objet Connection "Explicit Messaging"		
	Non-existant (inexistant)	Established (établie)	Deferred Delete (suppression différée).
Receive_Data	Non applicable	Si un message ou fragment de message valide a été reçu, réinitialiser le temporisateur Inactivity/Watchdog Timer ^b . Traiter/stocker le fragment ou bien gérer le message explicite.	Si un message ou fragment de message valide a été reçu, redémarrer le temporisateur Inactivity/Watchdog Timer ² et revenir à l'état Established . Traiter/stocker le fragment ou bien gérer le message explicite.
Send_Message	Non applicable	Examiner la longueur du message à émettre et, le cas échéant, accomplir une série fragmentée d'émissions. Autrement, émettre le message explicite complet.	Examiner la longueur du message à émettre et, le cas échéant, accomplir une série fragmentée d'émissions. Autrement, émettre le message explicite complet.
Le temporisateur Inactivity/Watchdog Timer expire.	Non applicable	Si l'attribut watchdog_timeout_action est mis à <i>Auto Delete</i> ou est mis à <i>Deferred Delete</i> et aucune instance de connexion enfant n'est dans l'état Established , libérer toutes les ressources associées et passer à Non-existent . Si l'attribut watchdog_timeout_action est mis à <i>Deferred Delete</i> et au moins une instance de connexion enfant est dans l'état Established , passer à Deferred Delete .	Non applicable
Instance de connexion enfant Deleted (supprimée) ou passage à Timed Out	Non applicable	Ignorer l'événement.	Si aucune instance de connexion enfant n'est dans l'état Established , libérer toutes les ressources associées et passer à Nonexistent .
^a Si la configuration indique qu'une ressource de connexion a besoin d'être allouée et il n'existe pas de ressource disponible, il est retourné une réponse d'erreur dont le Code d'erreur général indique "Resource Unavailable" (ressource non disponible) (02hex). ^b Sur CP 2/3, le champ MAC ID dans l'en-tête de message de tous les messages explicites et/ou fragments de message est examiné. Si l'ID de MAC de la destination est spécifiée dans l'ID de connexion (CAN Identifier Field), l'ID de MAC de source de l'autre point d'extrémité doit être spécifié dans l'en-tête de message. Si l'ID de MAC de la source est spécifiée dans l'ID de connexion, l'ID de MAC du module de réception doit être spécifié dans l'en-tête de message. Si l'une de ces vérifications échoue, le temporisateur Inactivity/Watchdog Timer n'est pas réinitialisé et le message/fragment de message est rejeté.			
NOTE Les Règles d'accès d'attribut sont spécifiées dans la CEI 61158-5-2:2010, paragraphe 6.2.3.2.1.4			

Important: L'événement Receive_Data n'est délivré à une connexion "Explicit Messaging " que lorsqu'il est reçu un message dont l'ID de connexion concorde avec l'attribut consumed_connection_id. S'il est reçu un message dont l'id de connexion ne concorde avec aucun attribut consumed_connection_id d'un objet Connection "Established", le message est rejeté.

Si une mise en œuvre détecte qu'elle ne prend pas en charge un service de messagerie explicite indiqué dans le Tableau 208, il est retourné une réponse d'erreur spécifiant *Service Not Supported* (Code de statut général 08).

8 Diagramme protocolaire de services de la FAL (FSPM)

8.1 Généralités

Ce diagramme d'états protocolaires de services FAL de bus de terrain met l'utilisateur de la FAL en correspondance avec les services d'objets de communication internes à la FAL.

8.2 Définitions des primitives

Les objets au sein du diagramme FSPM doivent fournir les services suivants:

Get_Attribute_All	Lit les valeurs de tous les attributs de la classe ou instance d'objet spécifiée.
Set_Attribute_All	Écrit dans tous les attributs de la classe ou instance d'objet spécifiée les valeurs spécifiées.
Get_Attribute_List	Lit les valeurs de la liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.
Set_Attribute_List	Écrit dans la liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée les valeurs spécifiées.
Get_Attribute_Single	Lit la valeur de l'attribut spécifié de la classe ou de l'instance d'objet spécifiée.
Set_Attribute_Single	Écrit dans l'attribut spécifié de la classe ou de l'instance d'objet spécifiée la valeur spécifiée.
Reset (réinitialisation)	Réinitialise la classe ou l'instance d'objet spécifiée
Create	Crée une instance de la classe d'objets spécifiée.
Delete (c'est-à-dire effacer, détruire)	Supprime une instance de la classe d'objets spécifiée.
Start	Démarre l'exécution de l'objet spécifié.
Stop	Arrête l'exécution de l'objet spécifié.
Find_Next_Object_Instance	Trouve l'identificateur de la prochaine instance non utilisée de la classe d'objets spécifiée.
NOP	Déclenche la réponse NOP correspondante issue de l'objet spécifié.
Apply_Attributes	Amène les valeurs d'attribut en cours à devenir actives dans l'objet spécifié.
Save	Sauvegarde le contenu des attributs de l'objet spécifié
Restore	Restaure le contenu des attributs de l'objet spécifié
Group_Sync	Vérifie que chaque membre d'un groupe est synchronisé au temps du système.

Pour les définitions détaillées de ces services, se référer à la CEI 61158-5-2:2010.

Les primitives des services communs échangés entre l'UCMM et l'utilisateur de la FAL sont montrées dans le Tableau 210 et dans le Tableau 211.

Tous les services ont les paramètres communs ci-après, montrés dans le Tableau 209.

Tableau 209 – Primitives émises par l'utilisateur de la FAL vers le diagramme FSPM

Paramètres communs	Req	Ind	Rsp	Cnf
Argument	M			
AREP	M			
Local	S			
UCMM Record identifier	S			
Transport identifier	S			
Receiver/Server Local ID		M		
Path	M	C		
Routing Path	M			
Add. Path	U	U(=)		
Port ID		M		
Result (+)			S	S(=)
AREP				M=
Receiver/Server Local ID			M=	
Service status			M	M(=)
Result (-)			S	S(=)
AREP				M=
Receiver/Server Local ID			M=	
Service status			M	M(=)

Seuls les paramètres arguments complémentaires des paramètres communs sont montrés dans le Tableau 210 et dans le Tableau 211.

Les paramètres complémentaires des primitives de services "indication" sont les même ceux de la primitive "request" correspondante.

Les paramètres complémentaires des primitives de services "confirmation" sont les même ceux de la primitive "response" correspondante.

Tableau 210 – Primitives émises par l'utilisateur de la FAL vers le diagramme FSPM

Nom de primitive	Source	Paramètres complémentaires	Fonctions
Get_attribute_all.req	Utilisateur de la FAL	Aucun	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour fournir des valeurs de tous les attributs de la classe ou de l'instance d'objet spécifiée.
Set_attribute_all.req	Utilisateur de la FAL	Attribute Data (Données d'attribut)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour écrire des valeurs spécifiées de tous les attributs de la classe ou de l'instance d'objet spécifiée.
Get_attribute_list.req	Utilisateur de la FAL	Attribute Count Attribute List	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour fournir des valeurs d'une liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.
Set_attribute_list.req	Utilisateur de la FAL	Attribute Count Attribute Data	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour écrire des valeurs spécifiées d'une liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.
Get_attribute_single.req	Utilisateur de la FAL	Aucun	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour fournir des valeurs de l'attribut spécifié de la classe ou de l'instance d'objet spécifiée.
Set_attribute_single.req	Utilisateur de la FAL	Attribute Data (Données d'attribut)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour écrire dans l'attribut spécifié de la classe ou de l'instance d'objet spécifiée les valeurs spécifiées.
Reset.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL pour réinitialiser la classe ou l'instance spécifiée de l'objet-cible.
Create.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour créer une instance de la classe d'objets spécifiée.
Delete.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour supprimer une instance de la classe d'objets spécifiée.
Start.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour démarrer une instance de la classe d'objets spécifiée.
Stop.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour arrêter une instance de la classe d'objets spécifiée.
Find_next_object_instance.req	Utilisateur de la FAL	Maximum Returned Values (valeurs maximales retournées)	Achemine une demande de l'utilisateur de la FAL vers un dispositif-cible pour trouver l'identificateur de la prochaine instance non utilisée de la classe d'objets spécifiée.
NOP.req	Utilisateur de la FAL	Aucun	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour renvoyer une réponse NOP correspondante.
Apply_Attributes.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour activer des valeurs d'attribut en cours.
Save.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour sauvegarder ses contenus d'attribut.
Restore.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour restaurer ses contenus d'attribut.

Nom de primitive	Source	Paramètres complémentaires	Fonctions
Group_Sync.req	Utilisateur de la FAL	Object Specific Data (facultatif)	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour vérifier que chaque membre d'un groupe est synchronisé au temps du système.
Get_attribute_all.rsp	Utilisateur de la FAL	(+) Attribute Data (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour fournir des valeurs de tous les attributs de la classe ou de l'instance d'objet spécifiée.
Set_attribute_all.rsp	Utilisateur de la FAL	(+) Aucun (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour écrire des valeurs spécifiées de tous les attributs de la classe ou de l'instance d'objet spécifiée.
Get_attribute_list.rsp	Utilisateur de la FAL	(+) Attribute Count Attribute Data (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour fournir des valeurs d'une liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.
Set_attribute_list.rsp	Utilisateur de la FAL	(+) Attribute_count Attribute Status List (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour écrire des valeurs spécifiées d'une liste spécifiée d'attributs de la classe ou de l'instance d'objet spécifiée.
Get_attribute_single.rsp	Utilisateur de la FAL	(+) Attribute Data (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour fournir des valeurs de l'attribut spécifié de la classe ou de l'instance d'objet spécifiée.
Set_attribute_single.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour écrire dans l'attribut spécifié de la classe ou de l'instance d'objet spécifiée les valeurs spécifiées.
Reset.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour confirmer que l'instance de la classe d'objets spécifiée a été réinitialisée.
Create.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour confirmer que l'instance de la classe d'objets spécifiée a été créée.
Delete.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour confirmer que l'instance de la classe d'objets spécifiée a été supprimée.
Start.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour confirmer que l'instance de la classe d'objets spécifiée a été démarrée.
Stop.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour confirmer que l'instance de la classe d'objets spécifiée a été arrêtée.
Find_next_object_instance.rsp	Utilisateur de la FAL	(+) Number Of List Members Instance ID List (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un dispositif-cible pour trouver l'identificateur de la prochaine instance non utilisée de la classe d'objets spécifiée.
NOP.rsp	Utilisateur de la FAL	Aucun	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour confirmer que le NOP a été reçu.
Apply_Attributes.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour confirmer que les valeurs d'attribut en cours ont été activées.
Save.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour confirmer que ses contenus d'attribut ont été sauvegardés.

Nom de primitive	Source	Paramètres complémentaires	Fonctions
Restore.rsp	Utilisateur de la FAL	(+) Object Specific Data (facultatif) (-) Specific Status Code	Retourne une réponse de l'utilisateur de la FAL vers un objet-cible pour confirmer que ses contenus d'attribut ont été restaurés.
Group_Sync.rsp	Utilisateur de la FAL	(+) IsSynchronized Object Specific Data (facultatif) (-) Specific Status Code	Achemine une demande de l'utilisateur de la FAL vers un objet-cible pour vérifier que chaque membre d'un groupe est synchronisé au temps du système.

Tableau 211 – Primitives émises par le diagramme FSPM vers l'utilisateur de la FAL

Nom de primitive	Source	Paramètres complémentaires	Fonctions
Get_attribute_all.ind	FSPM	idem que dans la primitive .req	Indique la réception de Get_attribute_all.req
Set_attribute_all.ind	FSPM	idem que dans la primitive .req	Indique la réception de Set_attribute_all.req
Get_attribute_list.ind	FSPM	idem que dans la primitive .req	Indique la réception de Get_attribute_list.req
Set_attribute_list.ind	FSPM	idem que dans la primitive .req	Indique la réception de Set_attribute_list.req
Get_attribute_single.ind	FSPM	idem que dans la primitive .req	Indique la réception de Get_attribute_single.req
Set_attribute_single.ind	FSPM	idem que dans la primitive .req	Indique la réception de Set_attribute_single.req
Reset.ind	FSPM	idem que dans la primitive .req	Indique la réception de Reset.req
Create.ind	FSPM	idem que dans la primitive .req	Indique la réception de Create.req
Delete.ind	FSPM	idem que dans la primitive .req	Indique la réception de Delete.req
Start.ind	FSPM	idem que dans la primitive .req	Indique la réception de Start.req
Stop.ind	FSPM	idem que dans la primitive .req	Indique la réception de Stop.req
Find_next_object_instance.ind	FSPM	idem que dans la primitive .req	Indique la réception de Find_next_object_instance.req
NOP.ind	FSPM	idem que dans la primitive .req	Indique la réception de NOP.req
Apply_Attributes.ind	FSPM	idem que dans la primitive .req	Indique la réception de Apply_Attributes.req
Save.ind	FSPM	idem que dans la primitive .req	Indique la réception de Save.req
Restore.ind	FSPM	idem que dans la primitive .req	Indique la réception de Restore.req
Group_Sync.ind	FSPM	idem que dans la primitive .req	Indique la réception de Group_Sync.req
Get_attribute_all.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Get_attribute_all.rsp
Set_attribute_all.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Set_attribute_all.rsp
Get_attribute_list.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Get_attribute_list.rsp

Nom de primitive	Source	Paramètres complémentaires	Fonctions
Set_attribute_list.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Set_attribute_list.rsp
Get_attribute_signle.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Get_attribute_single.rsp
Set_attribute_single.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Set_attribute_single.rsp
Reset.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Reset.rsp
Create.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Create.rsp
Delete.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Delete.rsp
Start.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Start.rsp
Stop.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Stop.rsp
Find_next_object_instance.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Find_next_object_instance.rsp
NOP.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de NOP.rsp
Apply_Attributes.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Apply_Attributes.rsp
Save.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Save.rsp
Restore.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Restore.rsp
Group_Sync.cnf	FSPM	idem que dans la primitive .rsp	Indique la réception de Group_Sync.rsp

8.3 Paramètres des primitives

Les paramètres utilisés avec les primitives échangées entre le diagramme FSPM et l'ARPM sont décrits dans le Tableau 212.

Tableau 212 – Paramètres utilisés avec les primitives échangées entre l'utilisateur de la FAL et le diagramme FSPM

Nom de paramètre	Description
AREP: Local identifier UCMM Record identifier Transport identifier	Identifie l'entité à utiliser pour acheminer le service. Ce paramètre peut utiliser un identificateur spécialisé associé à une entité locale, l'identificateur d'un enregistrement de transaction UCMM créé au préalable, ou l'identificateur de transport retourné par le processus d'établissement de connexion et associé à l'AR («Association Relationship» Relation d'applications) sélectionnée.
Receiver/Server Local ID	Généré par l'ASE Message Router du nœud répondeur. Identifie localement cette invocation du service. Il est utilisé pour associer des réponses de services à des indications.
Path: Routing Path Additional path	Dans la demande, il spécifie l'APO («Application Process Object» Objet de processus d'application) de la FAL ou l'élément d'APO de FAL qui est la destination de la demande de service. Dans l'indication, il contient seulement les segments au-delà du segment de classe logique issu de la demande de service, c'est-à-dire les informations complémentaires facultatives pour l'APO-cible (Add.Path).
Port ID	Indique sur quel port du dispositif l'indication de service est arrivée.
Service status: Status Code (obligatoire) Extended Status (conditionnel)	Donne des informations relatives au résultat d'exécution du service. Il est retourné dans toutes les primitives "confirmed service response" (+ et -). Status code (code de statut) indique le type d'erreur (voir la CEI 61158-5-2:2010 pour des détails). Extended status code (code de statut étendu) donne des détails du statut (voir la CEI 61158-5-2:2010 pour des détails).
Attribute Data (Données d'attribut)	1) Un train d'informations contenant tous les attributs. Les classes/objets qui prennent en charge ce service doivent définir le format de ce paramètre. 2) Une matrice de structures, prédéfinie par le système pour le service donné
Attribute Count (compte d'attributs)	Nombre de numéros d'attributs dans la liste d'attributs
Attribute list []	Liste (matrice) des numéros d'attribut des attributs à récupérer provenant de la classe ou de l'objet
Attribute Number	Numéro représentant la valeur de l'attribut
Attribute Status (statut d'attribut)	Information de statut d'attribut
Attribute Value (valeur d'attribut)	Séquence de données spécifiques à l'attribut de l'objet ou de la classe
Object Specific Data	Paramètres spécifiques à une classe/à une instance. La spécification de la classe/de l'instance doit spécifier le format.
Instance ID	Valeur attribuée pour identifier l'objet nouvellement créé
Maximum Returned Values (valeurs maximales retournées)	Nombre maximum de valeurs d'ID d'instance à retourner dans le message de réponse
Number Of List Members (nombre des éléments de liste)	Nombre des ID d'instance spécifiés dans le message de réponse
Instance ID List (liste des ID d'instance)	Liste des ID d'instance retournés. Les ID d'instance sont retournés dans des champs entiers de 16 bits.
IsSynchronized	Indique si l'objet est synchronisé à la Base de temps PTP

8.4 Diagrammes d'états FSPM

Le diagramme d'états FSPM n'a obtenu qu'un seul état possible: Running (en marche).

9 Diagrammes protocolaires de relation d'applications (ARPM)

9.1 Généralités

Le présent bus de terrain a des diagrammes protocolaires de relation d'applications pour:

- les relations d'applications sans connexion,
- les relations d'applications orientées connexion.

La relation sans connexion est d'un seul type uniquement.

Les relations orientées connexion s'inscrivent dans l'une de 7 classes de transport:

- 0 Null (ou Base)
- 1 Duplicate Detection (détection de doublons)
- 2 Acknowledged (acquitté)
- 3 Verified (vérifié)
- 4 Non-blocking (non bloquant)
- 5 Non-blocking, Fragmenting (Non bloquant, fragmentant)
- 6 Multipoint, Fragmenting (multipoint, fragmentant)

Bien que similaires, ces transports ont chacun leur propre ARPM.

9.2 ARPM sans connexion (UCMM)

9.2.1 Généralités

Les fonctions des ARPM sans connexion sont fournies par l'objet Unconnected Message Manager (UCMM, gestionnaire de messages non connectés). L'UCMM doit fournir des services de messages de demande/réponse non connectés, limités à une seule liaison qui prend en charge plusieurs messages en cours. Le nombre requis de messages en cours doit être spécifique à une mise en œuvre. L'UCMM doit être présent dans tous les nœuds et doit prendre en charge au moins un message en cours.

9.2.2 Définitions des primitives

L'objet UCMM doit fournir les services suivants:

UCMM_Create	Crée une instance d'enregistrement de transaction. Place l'UCMM dans l'état "Running". Ce service est local, il n'entraîne pas l'envoi d'une PDU.
UCMM_Delete	Supprime un enregistrement de transaction existant. Place l'UCMM dans l'état "Inactive" (inactif). Ce service est local, il n'entraîne pas l'envoi d'une PDU. Ce service est local, il n'entraîne pas l'envoi d'une PDU.
UCMM_Write	Écrit un élément de données d'application dans le tampon des PDU de transport; cela se traduit par l'envoi des données vers une cible spécifiée.
UCMM_Abort	Abandonne une transaction existante.

Pour les définitions détaillées de ces services, se référer à la CEI 61158-5-2:2010.

Les primitives échangées entre l'UCMM et le diagramme FSPM sont montrées dans le Tableau 213 et dans le Tableau 214 ci-après.

Tableau 213 – Primitives émises par le diagramme FSPM vers l'ARPM

Nom de primitive	Source	Paramètres associés	Fonctions
UCMM_Create_req	FSPM	fixed tag max retries,	Achemine une demande du diagramme FSPM vers l'ARPM pour créer un enregistrement de transaction.
UCMM_Delete_req	FSPM	record	Achemine une demande du diagramme FSPM vers l'ARPM pour supprimer un enregistrement de transaction établi précédemment.
UCMM_Write_req	FSPM	record destination ID UCMM service response timeout transaction priority application data	Achemine une demande du diagramme FSPM vers l'ARPM pour envoyer un élément de données d'application en utilisant un enregistrement de transaction créé précédemment.
UCMM_Write_rsp	FSPM	record application data	Achemine une réponse du diagramme FSPM, en passant par un routeur de message (Message Router), vers l'ARPM pour envoyer une réponse à UCMM_Send_req en utilisant un enregistrement de transaction créé précédemment.
UCMM_Abort_req	FSPM	record	Abandonne la transaction correspondant à l'enregistrement de transaction spécifié; retire le paquet de la DLL locale s'il n'a pas encore été émis.

Tableau 214 – Primitives émises par de l'ARPM vers le diagramme FSPM

Nom de primitive	Source	Paramètres associés	Fonctions
UCMM_Create_cnf	ARPM	record service_status	Achemine de l'ARPM vers le FSPM une confirmation que l'enregistrement de transaction a été créé.
UCMM_Write_ind	ARPM	record source ID application data	Achemine de l'ARPM vers le Message Router une indication que des données sont arrivées sur l'enregistrement de transaction créé précédemment. Le Message Router transmet alors l'indication à l'objet d'application approprié.
UCMM_Write_cnf	ARPM	record service status application data	Achemine de l'ARPM vers le routeur de message (Message Router) une confirmation de l'exécution de UCMM_Send_req du FSPM sur un enregistrement de transaction créé précédemment.

9.2.3 Paramètres des primitives

Les paramètres utilisés avec les primitives échangées entre le diagramme FSPM et l'ARPM sont décrits dans le Tableau 215.

Tableau 215 – Paramètres utilisés avec les primitives échangées entre le diagramme FSPM et l'ARPM

Nom de paramètre	Description
record	Identifie sans ambiguïté l'instance de la connexion temporaire sur laquelle le diagramme FSPM a émis une primitive "request" UCMM_Send. Un moyen pour une telle identification n'est pas spécifié dans la présente norme.
application data	Achemine des données d'utilisateur de la FAL.
fixed tag	Mis à la valeur de 0x83 pour l'UCMM et de 0x88 pour UCMM de gestion (Management UCMM)
destination ID	Identificateur non ambigu (ID de MAC) du nœud-source de l'UCMM_write_req.
source ID	Identificateur non ambigu (ID de MAC) du nœud-destination de l'UCMM_write_req.
service	Le paramètre "service" doit spécifier le mécanisme de livraison à utiliser pour ce message et doit être l'un choisi parmi: Request_With_Acknowledge = 2 (répétitions de tentatives jusqu'à l'acquittement) Request_With_No_Response = 4 (pas d'acquittement) Request_With_No_ACK = 7 (répétitions de tentatives jusqu'à la réponse) Request_With_No_Retry = 8 (avec réponse) NOTE Ces valeurs correspondent à des codes de commandes contenus dans l'en-tête UCMM.
timeout	Durée de la transaction en microsecondes. Si aucun UCMM_Send_cnf n'est reçu avant que la temporisation expire, la transaction est abandonnée et le paramètre send_status doit être TIMEOUT.
retries	Achemine le nombre maximal autorisé de répétitions de tentative.
create_status	Statut retourné avec UCMM_Create_conf = - success (succès) - cannot create (ne peut pas créer)
send_status	Statut retourné avec UCMM_Send_conf = - success - record_not_created - packet_too_big - no_acknowledge - aborted - timeout

9.2.4 Diagrammes d'états UCMM

9.2.4.1 Client UCMM

9.2.4.1.1 États

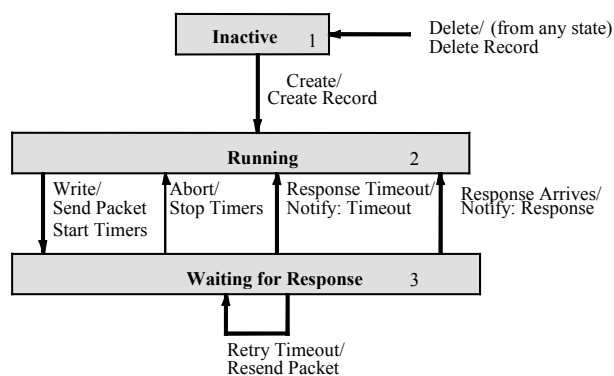
Les états définis et leurs descriptions du client UCMM sont énumérés dans le Tableau 216.

Tableau 216 – États du client UCMM

État	Description
Inactive (Inactif)	L'enregistrement pour le transfert d'UCMM n'existe pas.
Running (En marche)	L'enregistrement pour le transfert d'UCMM a été créé.
Waiting for response (En attente de réponse)	Le client a envoyé un élément de données d'application et attend une réponse.

9.2.4.1.2 Diagramme de transitions d'états

La Figure 28 montre le diagramme de transitions d'états du client UCMM.



Légende

Anglais	Français
Delete/ (from any state)	Supprimer (de n'importe quel état)
Delete Record	Supprimer enregistrement
Inactive	Inactif
Create/Create Record	Créer/Créer un enregistrement
Running	En marche
Write/ Send Packet	Écrire/ envoyer le paquet
Start Timers	Démarrer les temporisateurs
Abort/ Stop Timers	Abandonner/ Arrêter les temporisateurs
Response Timeout/Notify: Timeout	Temporisation de réponse/ Notifier: temporisation
Response Arrives/ Notify: Response	Réponse arrive/ Notifier: temporisation
Waiting for Response	Attente d'une réponse
Retry Timeout/Resend Packet	Temporisation de répétition/ Envoyer de nouveau le paquet

Figure 28 – Diagramme de transitions d'états de client UCMM

9.2.4.1.3 Matrice d'événements d'états

Les transitions d'états pour le client UCMM doivent être telles que spécifiées dans le Tableau 217.

Tableau 217 – Matrice d'événements d'états du client UCMM

Événement	État		
	Inactive	Running	Waiting for response
Create.req	1) créer enregistrement 2) passer à Running		
Delete.req	aucune action	1) supprimer enregistrement 2) incrémenter SEQUENCE 3) passer à Inactive	1) supprimer enregistrement 2) incrémenter SEQUENCE 3) passer à Inactive
Write.req	confirmer status = INVALID_RECORD	1) envoyer paquet 2) démarrer Temporisateur de réponse 3) Réinitialiser le compte de répétitions de tentative 4) Démarrer le temporisateur de répétitions de tentative 5) passer à Waiting	confirm status = BUSY
Abort.req	confirmer status = INVALID_RECORD	aucune action	1) arrêter les temporisateurs 2) incrémenter SEQUENCE 3) passer à Inactive
Temporisation de répétitions de tentative Paquet de demande disponible			1) Décrémenter le compte de répétitions de tentative SI le compte de répétitions de tentative a expiré 2) Notifier: Temporisation - pas d'ACK 3) Libérer tampon de demandes 4) Incrémenter Sequence n° 5) Arrêter temporisateur de réponse 6) Passer à Running SINON. 1) Renvoyer paquet 2) Redémarrage temporisateur de répétitions de tentative
Temporisation réponse			1) confirmer avec = TIMEOUT-pas de réponse 2) incrémenter SEQUENCE 3) passer à Running
Réponse arrive		aucune action	1) confirmer avec statut = SUCCESS 2) Envoyer ACK_RESP, sauf si réponse indique aucune ACK_RESP souhaitée 3) incrémenter SEQUENCE 4) passer à Running
ACK_REQ arrive, le numéro de séquence correspond à la valeur stockée.		Aucune action	1) Libérer tampon de demandes 2) arrêter le temporisateur de répétitions de tentative
ACK_REQ arrive, le numéro de séquence n'est pas égal à la valeur stockée.		Aucune action	Aucune action

Le numéro de séquence doit être mis à zéro à l'initialisation. La valeur du numéro de séquence doit être maintenue dans l'état "inactif", à utiliser lorsque record (l'enregistrement) passe à Running. La valeur de temporisation des répétitions de tentative doit être fixe pour la liaison à une valeur qui garantit que les nœuds tant clients que serveurs ont une opportunité d'émettre un paquet non programmé.

La temporisation de réponse est le temps de réponse fourni lorsque l'enregistrement est créé.

9.2.4.2 Serveur UCMM haut de gamme

9.2.4.2.1 Fonctions

Lorsqu'un paquet arrive au serveur, une instance de transaction doit être créée si les ressources sont disponibles. Si les ressources ne sont pas disponibles, le paquet doit être rejeté. Lorsque l'instance est créée, un enregistrement de transaction doit être créé pour l'instance en question. Cet enregistrement doit être actif pour la durée de vie de cette transaction. La transaction doit finir:

- lorsqu'un ACK_RESP est reçu après qu'une réponse a été envoyée;
- lorsque le temporisateur de temps de réponse expire; ou
- lorsqu'un nouveau paquet est reçu après qu'une réponse a été envoyée.

9.2.4.2.2 États

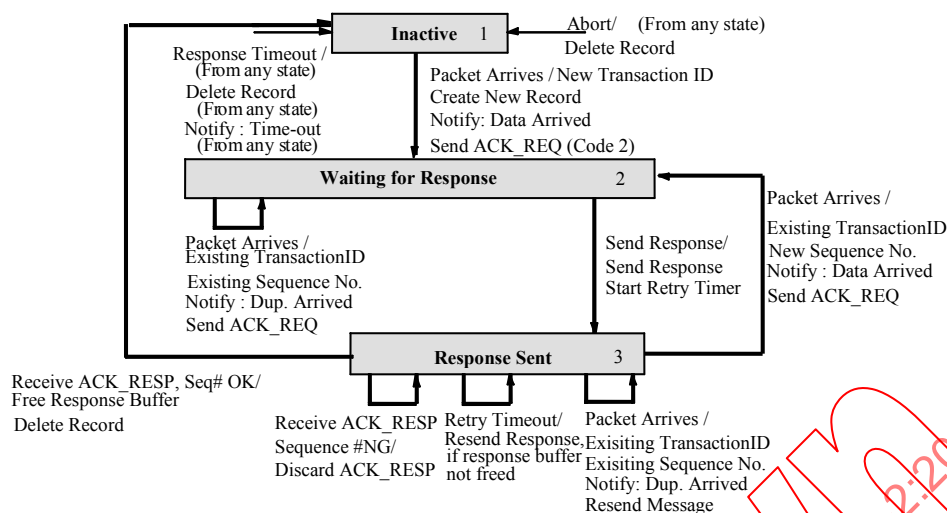
Les états définis et leurs descriptions du serveur UCMM haut de gamme sont énumérés dans le Tableau 218.

Tableau 218 – États du serveur UCMM haut de gamme

État	Description
Inactive	L'enregistrement pour le transfert d'UCMM n'existe pas.
Waiting for response (En attente de réponse)	Un paquet de données est arrivé avec un nouvel ID de transaction. Un nouvel enregistrement est créé. Le serveur attend que la réponse soit envoyée par l'application serveur.
Response sent (réponse envoyée)	L'application serveur a envoyé une réponse.

9.2.4.2.3 Transitions d'états

La Figure 29 montre le diagramme de transitions d'états pour un serveur UCMM haut de gamme.



Légende

Anglais	Français
Abort/ (From any state)	Abandonner/ (de n'importe quel état)
Delete Record	Supprimer l'enregistrement
Inactive	Inactif
Packet Arrives /	Paquet arrive/
New Transaction ID	Nouvel ID transaction
Create New Record	Créer nouvel enregistrement
Notify: Data Arrived	Notifier: Données arrivées
Send ACK_REQ (Code 2)	Envoyer ACK_REQ (code 2)
Response Timeout /	Temporisation de réponse
(From any state)	(de n'importe quel état)
Delete Record	Supprimer l'enregistrement
(From any state)	(de n'importe quel état)
Notify: Time-out	Notifier: Temporisation
(From any state)	(de n'importe quel état)
Waiting for Response	En attente de réponse
Packet Arrives /	Paquet arrive/
Existing TransactionID	ID de transaction existant
New Sequence No.	Nouveau n° de séquence
Notify: Data Arrived	Notifier: Données arrivées
Send ACK_REQ	Envoyer ACK_REQ
Send Response/	Envoyer réponse/
Send Response	Envoyer réponse
Start Retry Timer	Démarrer temporisateur de répétitions
Packet Arrives /	Paquet arrive/
Existing TransactionID	ID de transaction existant
Existing Sequence No.	N° séquence existant
Notify: Dup Arrived	Notifier: Doublet arrivé

Anglais	Français
Send ACK_REQ	Envoyer ACK_REQ
Response Sent	Réponse envoyée
Packet Arrives /	Paquet arrive/
Existing TransactionID	ID de transaction existant
Existing Sequence No.	N° séquence existant
Packet Arrives /	Paquet arrive/
Notify: Dup. Arrived	Notifier: Doublet arrivé
Resend Message	Envoyer de nouveau le message
Retry Timeout/	Temporisation de répétitions
Resend Response, if response buffer not freed	Envoyer de nouveau la réponse, si le tampon des réponses pas libre
Receive ACK_RESP	Recevoir ACK_RESP
Sequence #NG/	N° sequence NG/
Discard ACK_RESP	Rejeter ACK_RESP
Receive ACK_RESP, Seq# OK/	Recevoir ACK_RESP, n° Seq OK/
Free Response Buffer	Libérer le tampon des réponses
Delete Record	Supprimer l'enregistrement

Figure 29 – Diagramme de transitions d'états d'un serveur UCMM haut de gamme

9.2.4.2.4 Matrice d'événements d'états

Les transitions d'états pour le serveur UCMM haut de gamme doivent être telles que spécifiées dans le Tableau 219.

Le numéro de séquence doit être mis à zéro à l'initialisation. La valeur du numéro de séquence doit être maintenue dans l'état "Inactive", à utiliser lorsque record (l'enregistrement) passe à Running. La valeur de temporisation des répétitions de tentative doit être fixée pour la liaison locale.

La temporisation de réponse doit être le temps de réponse fourni lorsque l'enregistrement est créé.

Tableau 219 – Matrice d'événements d'états du serveur UCMM haut de gamme

Événement	État		
	Inactive	Waiting for response	Response sent (réponse envoyée)
Le paquet arrive demande (code 2) Nouvel ID de transaction et adresse source	1) Créer un nouvel enregistrement 2) Notifier: Données arrivées 3) Envoyer ACK_REQ 4) Démarrer le temporisateur de réponse 5) Passer à Waiting for App Response	Non applicable	Non applicable
Le paquet arrive demande (code 7) Nouvel ID de transaction et nouvelle adresse source	1) Créer un nouvel enregistrement 2) Notifier: Données arrivées 3) Démarrer le temporisateur de réponse 4) Passer à Waiting for App Response	Non applicable	Non applicable
Le paquet arrive. ID de transaction et adresse source existants Nouveau n° de séquence	Non applicable	Aucune action	1) Mettre à jour l'enregistrement 2) Notifier: Données arrivées 3) Envoyer ACK_REQ 4) Passer à Waiting for App Response
Le paquet arrive. ID de transaction existant N° de séquence existant	Non applicable	1) Notifier: Doublet Arrivé (Facultatif) 2) Envoyer Send ACK_REQ	1) Notifier: Doublet Arrivé (Facultatif) 2) Renvoyer message de réponse
Temporisateur de réponse expire.	Non applicable	1) Notifier: Temporisation 2) Supprimer l'enregistrement 3) Passer à "Inactive".	1) Notifier: Temporisation 2) Supprimer l'enregistrement 3) Passer à "Inactive".
Abandonner	Erreur	1) Supprimer l'enregistrement. 2) Passer à "Inactive".	1) Supprimer l'enregistrement. 2) Passer à "Inactive".
Envoyer réponse.	Erreur	1) Envoyer la réponse. 2) Démarrer le temporisateur de répétitions de tentative 3) Passer à Response Sent	Erreur

Événement	État		
	Inactive	Waiting for response	Response sent (réponse envoyée)
Temporisation des répétitions de tentative	Non applicable	Non applicable	1) Décrémenter le compte de répétitions de tentative SI le compteur de répétitions de tentative a expiré. 2) Libérer le tampon de réponses; 3) Arrêter le temporisateur de réponse; 4) Passer à "Inactive" SINON 1) Renvoyer le message de réponse; 2) Démarrer le temporisateur de répétitions de tentative.
ACK_RESP arrive, le numéro de séquence correspond à la valeur stockée.	Aucune action	Aucune action	1) Supprimer l'enregistrement. 2) Passer à "Inactive".
ACK_RESP arrive, le numéro de séquence n'est pas égal à la valeur stockée.	Aucune action	Aucune action	Aucune action

Le temps de réponse est le temps de réponse reçu dans l'en-tête de message.

9.2.4.3 Serveur UCMM bas de gamme

9.2.4.3.1 Fonctions

Le serveur bas de gamme ne peut pas prendre en charge les fonctions suivantes:

- accomplir des répétitions de tentative de message de réponse,
- détecter les doublons de messages.

9.2.4.3.2 États

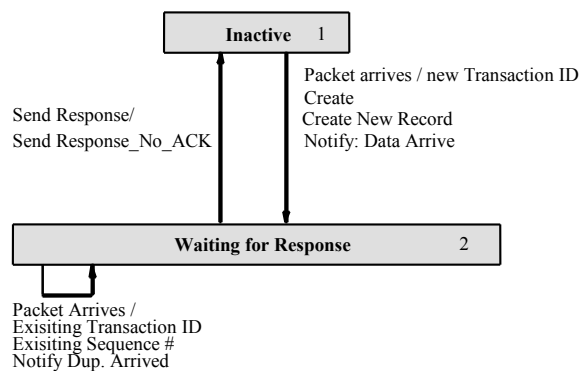
Les états définis et leurs descriptions du serveur UCMM bas de gamme sont énumérés dans le Tableau 220.

Tableau 220 – États du serveur UCMM bas de gamme

État	Description
Inactive	L'enregistrement pour le transfert d'UCMM n'existe pas.
Waiting for response	Un paquet de données est arrivé avec un nouvel ID de transaction. Un nouvel enregistrement est créé. Le serveur attend que la réponse soit envoyée par l'application serveur.

9.2.4.3.3 Transitions d'états

La Figure 30 montre le diagramme de transitions d'états pour un serveur UCMM bas de gamme.

**Légende**

Anglais	Français
Inactive	Inactif
Packet arrives / new Transaction ID	Le paquet arrive/ nouvel ID de transaction
Create	Créer
Create New Record	Créer un nouvel enregistrement
Notify: Data Arrive	Notifier: Données arrivées
Send Response/	Envoyer réponse/
Send Response_No_ACK	Envoyer
Waiting for Response	En attente de réponse
Packet Arrives /	Le paquet arrive/
Existing Transaction ID	ID de transaction existant
Existing Sequence #	Séquence existante n°
Notify Dup. Arrived	Notifier: Doublet arrivé

Figure 30 – Diagramme de transitions d'états pour serveur UCMM bas de gamme**9.2.4.3.4 Matrice d'événements d'états**

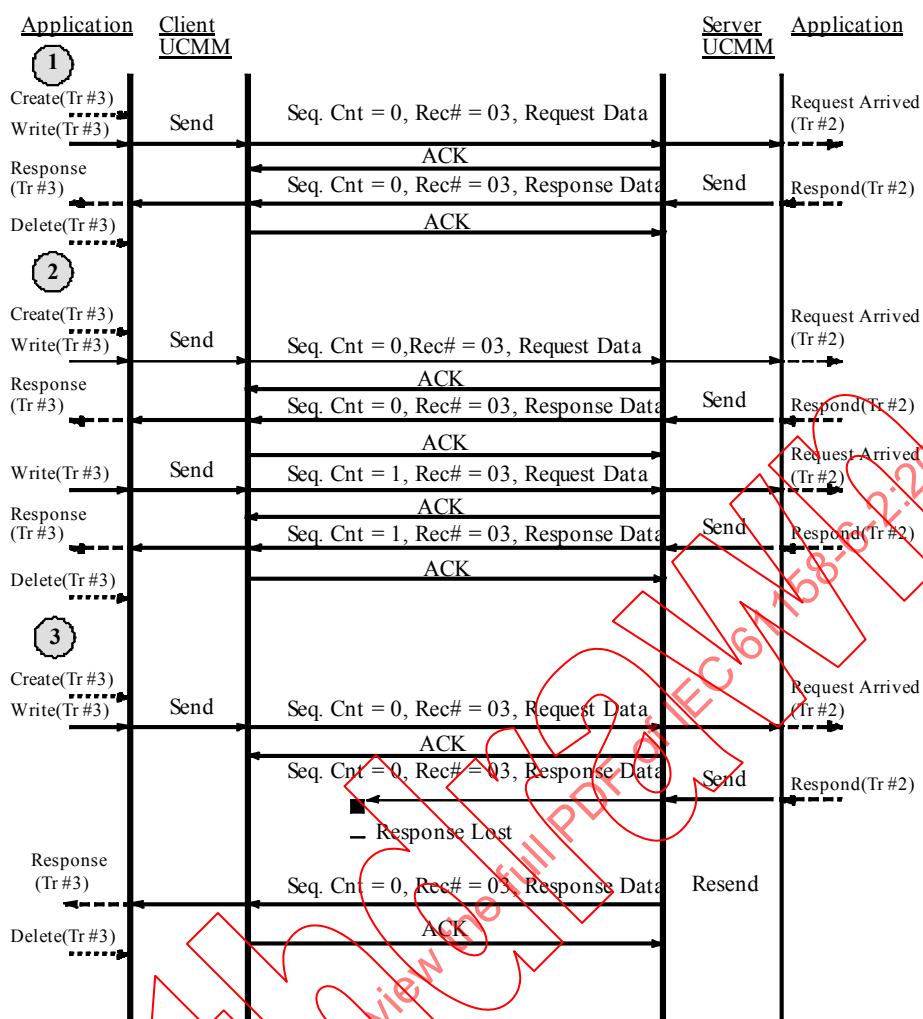
Les transitions d'états pour le serveur UCMM bas de gamme doivent être telles que spécifiées dans le Tableau 221.

Tableau 221 – Matrice d'événements d'états du serveur UCMM bas de gamme

Événement	État	
	Inactive	Waiting for response
Le paquet arrive. Nouvel ID de transaction et nouvelle adresse source	1) Créer un nouvel enregistrement 2) Notifier: Données arrivées 3) Si réponse pas disponible immédiatement, Envoyer ACK_REQ 4) Passer à Waiting for App Response	Non applicable
Le paquet arrive. ID de transaction et adresse source existants Nouveau n° de séquence	Aucune action	Aucune action
Le paquet arrive. ID de transaction existant N° de séquence existant	Aucune action	1) Notifier: Doublet arrivé. (Facultatif)
Envoyer réponse.	Erreur	1) Envoyer Response_No_ACK 2) Passer à "Inactive".
ACK_RESP arrive	Aucune action	Aucune action

9.2.5 Exemples de séquences UCMM

La Figure 31 montre un diagramme de séquence pour un UCMM avec un seul message en cours tandis que la Figure 32 montre un diagramme de séquence pour un UCMM avec plusieurs messages en cours.



Légende

Anglais	Français
Application	Application
Client UCMM	Client UCMM
Server UCMM	Server UCMM
Write(Tr #)	Écrire(Tr n°)
Request Arrived (Tr #)	Demande arrivée (Tr n°)
Seq. Cnt = 0, Rec# = 03, Request Data	Seq. Cnt = 0, Rec n° = 03, Données de demande
Send	Envoyer
Resend	Renvoyer
Request	Demande
Response	Réponse
Request Data	Données de demande
Response Data	Données de réponse
Respond(Tr #)	Répondre(Tr n°)
Delete(Tr #3)	Supprimer(Tr n°3)
Create(Tr #3)	Créer(Tr n°3)
ACK	«acquittement»
Rec #	Réception n°
Response Lost	Réponse perdue

Figure 31 – Diagramme de séquence pour un UCMM avec un seul message en cours

Exemple 1. Transaction typique: Une instance est créée, la demande est écrite, la réponse est retournée et l'instance est supprimée.

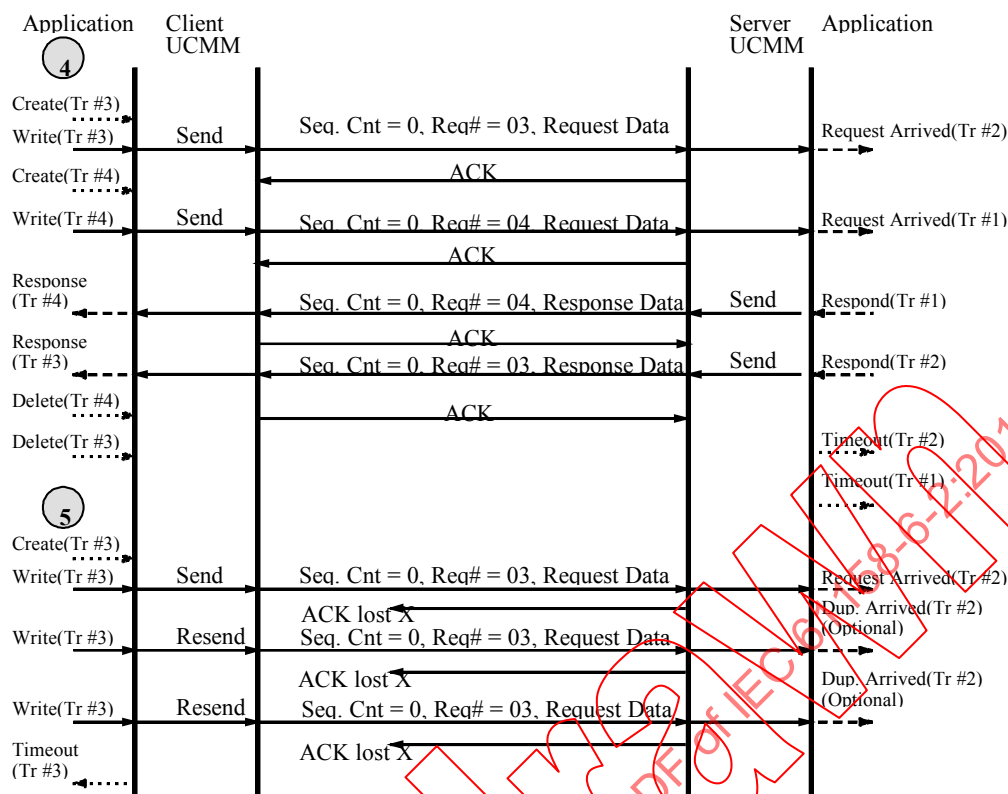
Exemple 2. Deux transactions/même instance: Dans ce cas, l'instance est ouverte et deux demandes sont envoyées.

NOTE 1 Le client attend la première réponse à retourner avant d'émettre une seconde demande en utilisant le même numéro de transaction.

Exemple 3. Réponse perdue: Dans ce cas, la réponse est perdue et le service émet une répétition de tentative.

IECNORM.COM: Click to view the full PDF of IEC 61158-6-2:2010

Without watermark



Légende

Anglais	Français
Application	Application
Client UCMM	Client UCMM
Server UCMM	Server UCMM
Write(Tr #	Écrire(Tr n°
Request Arrived	Demande arrivée
Dup. Arrived(Tr #	Doublon arrivé(Tr n°
Time out(Tr #	Temporisation(Tr n°
(Tr #	(Tr n°
Seq. Cnt = 0, Rec# = 03, Request Data	Seq. Cnt = 0, Rec n° = 03, Données de demande
Send	Envoyer
Resend	Renvoyer
Request	Demande
Response	Réponse
Request Data	Données de demande
Response Data	Données de réponse
Respond(Tr #)	Répondre(Tr n°)
Delete(Tr #	Supprimer(Tr n°
Create(Tr #	Créer(Tr n°
ACK	«acquiescement»
Req #	Demande n°
ACK Lost	Acquiescement perdu
Timeout	Temporisation
Optional	Facultatif

Figure 32 – Diagramme de séquence pour un UCMM avec plusieurs messages en cours

Exemple 4. Plusieurs en cours: Dans ce cas, le client crée deux instances et envoie une seconde demande avant que la première demande soit achevée.

NOTE 2 Le client utilise un numéro de transaction différent pour la seconde demande.

Exemple 5. Temporisation: Dans ce cas, l'application serveur ne répond pas et le client émet une temporisation. La temporisation se traduit par le retour d'un statut d'erreur vers l'application. L'instance de transaction doit être supprimée.

9.2.6 UCMM de gestion

Tout dispositif qui a mis en œuvre l'objet Keeper doit aussi mettre en œuvre le serveur Management UCMM (UCMM de gestion). À deux exceptions près, le serveur Management UCMM doit être identique au serveur UCMM à étiquette fixe 0x83 UCMM:

- les transactions du serveur Management UCMM doivent être émises sur l'étiquette fixe 0x88;
- le serveur Management UCMM ne doit émettre aucun paquet si son nœud ne contient pas un objet Keeper dans l'état de maître.

Un dispositif peut mettre en œuvre le client Management UCMM. Le client Management UCMM est identique au client UCMM à étiquette fixe 0x83, excepté que les transactions pour le client Management UCMM doivent être émises sur l'étiquette fixe 0x88.

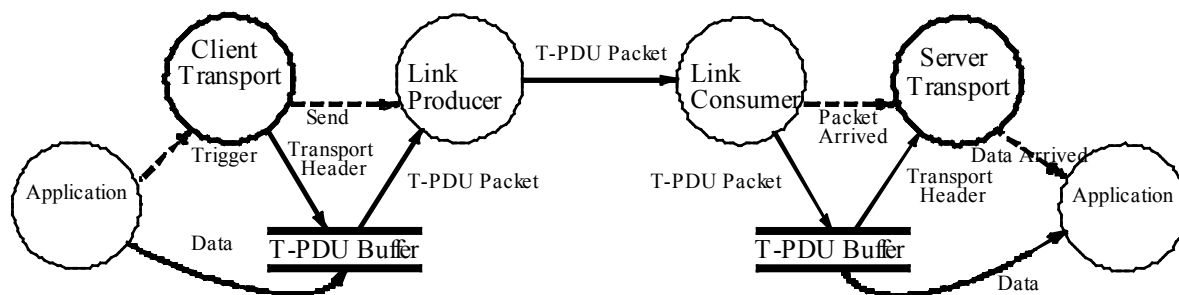
NOTE Les messages à l'objet Keeper sont diffusés, car tous les dispositifs sont autorisés à mettre en œuvre cet objet. Pour faciliter les communications vers l'objet Keeper sans avoir d'impact sur les dispositifs autres que Keeper, ces messages sont émis sur une étiquette fixe différente.

9.3 Diagrammes ARP orientés connexion (transports)

9.3.1 Tampon des PDU de transport

9.3.1.1 Format des PDU de transport

Le tampon de TPDU contient un paquet TPDU. Le paquet TPDU est constitué d'un en-tête et de données de transport. Les applications doivent écrire les données dans les tampons TPDU et en lire des données, et ce, directement. Les instances de transports doivent écrire et lire l'en-tête de transport dans le tampon TPDU alors que les consommateurs doivent écrire les données dans le tampon TPDU et les producteurs doivent lire les données dans le tampon TPDU. Les tampons TPDU et la gestion des tampons ne font pas partie des services de communication. Ce processus est montré à la Figure 33.



Légende

Anglais	Français
Application	Application
Client Transport	Transport Client
Trigger	Trigger
Data	Données
Send	Envoyer
Transport Header	En-tête de transport
T-PDU Buffer	Tampon des T-PDU
T-PDU Packet	Paquet de T-PDU
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Data Arrived	Données arrivées
Server Transport	Transport serveur
Packet arrived	Paquet arrivé

Figure 33 – Tampon des T-PDU

9.3.1.2 Gestion des tampons de PDU de transport

La gestion des tampons de T-PDU est spécifique à une mise en œuvre. Les mises en œuvre qui utilisent une seule mise en tampon, la double mise en tampon, l'écrasement en écriture ou la mise en file d'attente pour gérer les tampons de T-PDU sont toutes autorisées.

Les réalisateurs ont la responsabilité d'assurer l'intégrité des tampons (accès atomique).

Les applications ont la responsabilité d'assurer que le tampon a été initialisé avant qu'un déclencheur ne soit émis. L'échec à initialiser le tampon doit permettre l'envoi de données inconnues. Cela est vrai pour les producteurs et les côtés client et serveur.

9.3.1.3 Notification

Les classes de transport peuvent signaler l'application par l biais d'événements. Les événements valides sont tels que spécifiés dans le Tableau 222.

Tableau 222 – Notification

Événement	Client				Serveur			
	Classe 0	Classe 1	Classe 2	Classe 3	Classe 0	Classe 1	Classe 2	Classe 3
Données arrivées					n	n	n	n
Doublon arrivé						n	n	n
Acquittement			n					
Vérification				n				

NOTE Un événement *Doublon arrivé* ne présente normalement pas d'intérêt pour l'application; toutefois, il pourrait être utile pour déclencher le temporisateur de chien de garde afin d'indiquer que l'application client est active et assure un déclenchement.

9.3.2 Classes de transport

Les applications s'interfaçent aux services de transport par le truchement des classes de transport prises en charge. Le Tableau 223 énumère les classes de transport d'usage général qui ont été définies pour les systèmes. Chaque classe de transport fournit un niveau différent de fonctionnalité. La classe de transport 0 fournit la fonctionnalité minimale requise d'une classe de transport, permettant le transfert de données entre les applications.

Tableau 223 – Classes de transport

Numéro de classe	Nom de la classe
0	Null (ou Base)
1	Duplicate Detection («détection de doublons»)
2	Acknowledged (« acquitté »)
3	Verified (« vérifié »)
4	Non-blocking (« non bloquant »)
5	Non-blocking, Fragmenting (« Non bloquant, fragmentant »)
6	Multipoint, Fragmenting (« multipoint, fragmentant »)

9.3.3 Définitions des primitives communes

Les classes de transport doivent fournir les services suivants:

TR_Write	Écrit un élément de données d'application dans le tampon des PDU de transport d'envoi, prêt à être envoyé par le biais d'une connexion préétablie.
TR_Trigger	Fait envoyer par le truchement de la connexion préétablie un élément de données d'application ou de données de réponse placé au préalable dans le tampon des PDU de transport d'envoi.
TR_Packet_arrived	Indique à l'instance de transport qu'un paquet de données est arrivé dans le tampon des TPDU de transport de réception. Ce service est déclenché par une action locale.
TR_Ack_received	Indique à l'utilisateur qu'un paquet de données est arrivé dans le tampon des PDU de transport de réception signalant un acquittement de l'arrivée de données précédemment envoyées dans le tampon des PDU de transport de réception. Ce service est déclenché par une action locale.
TR_Verify	Indique à l'utilisateur qu'un paquet de données est arrivé dans le tampon des PDU de transport de réception signalant une vérification de l'arrivée de données précédemment

envoyées dans l'objet d'utilisateur de réception. Ce service est déclenché par une action locale.

TR_Status_Update Indique à l'utilisateur qu'un statut mis à jour est présent dans le tampon des PDU de transport.

Pour les définitions détaillées de ces services, se référer à la CEI 61158-5-2:2010.

Les primitives échangées entre les instances de transport et le diagramme FSPM sont montrées dans le Tableau 224 et dans le Tableau 225 ci-après.

Tableau 224 – Primitives émises par le diagramme FSPM vers l'ARPM

Nom de primitive	Source	Paramètres associés	Fonctions
TR_Write_req	FSPM	Transport identifier Application data	Achemine une demande du diagramme FSPM vers l'ARPM pour écrire des données d'application dans le tampon des PDU de transport.
TR_Trigger.req	FSPM	Transport identifier	Achemine une demande du diagramme FSPM vers l'ARPM pour déclencher le transfert de données d'application du tampon des PDU de transport vers le Producteur de liaison.
TR_Verify.req	FSPM	Transport identifier	Achemine une demande du diagramme FSPM vers l'ARPM pour déclencher le transfert de données de vérification d'écriture vers le tampon des PDU de transport.

Tableau 225 – Primitives émises par de l'ARPM vers le diagramme FSPM

Nom de primitive	Source	Paramètres associés	Fonctions
TR_Packet_arrived.ind	ARPM	Transport identifier Data arrived Duplicate arrived	Achemine une indication au FSPM qu'un élément de données est arrivé au consommateur de liaison.
TR_Ack_arrived.ind	ARPM	Transport identifier	Achemine une indication au FSPM qu'un acquittement est arrivé au consommateur de liaison.
TR_Status_Update.ind	ARPM	Transport identifier	Achemine une indication au FSPM qu'un statut nouveau est arrivé au consommateur de liaison.
TR_Verify.cnf	ARPM	Transport identifier	Achemine une confirmation de l'ARPM vers le FSPM pour indiquer la réception d'une DLPDU de vérification.

9.3.4 Paramètres des primitives communes

Les paramètres utilisés avec les primitives échangées entre le diagramme FSPM et l'ARPM sont décrits dans le Tableau 226.

Tableau 226 – Paramètres utilisés avec les primitives échangées entre le diagramme FSPM et l'ARPM

Nom de paramètre	Description
Transport identifier	Identificateur de l'instance du transport invoqué dans la transaction
Application data	Contient des données d'application (paramètres de demande/réponse de service ou données formatées d'utilisateur.
Data arrived Duplicate arrived	Les paramètres Data_Arrived et Duplicate_Arrived indiquent si le nouveau paquet contient de nouvelles données ou s'il est juste un doublon provenant d'un paquet reçu précédemment.

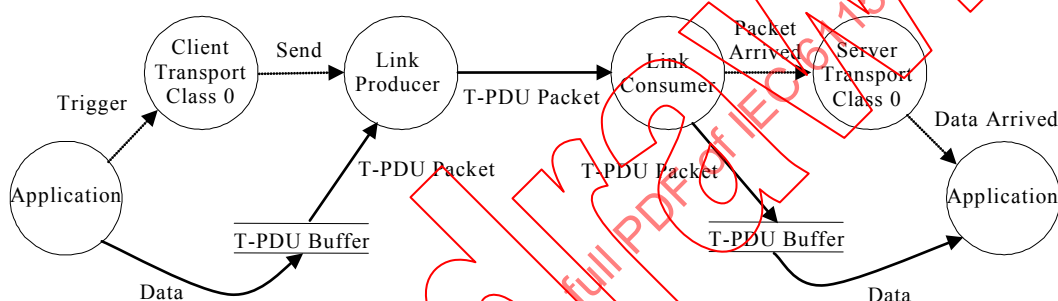
9.3.5 Diagrammes d'états de transport – classe 0

9.3.5.1 Fonctions

La classe de transport 0 est la classe de transport la plus simple et peut utiliser une connexion point à point ou multipoint. La classe 0 est typiquement utilisée pour émettre ou recevoir des entrées sur une connexion multipoint ou des sorties sur une connexion point à point.

Les utilisations possibles de la classe de transport 0 comprennent les entrées échantillonnées, les sorties normalisées, les transferts de blocs cycliques, les événements de diagnostic et la communication entre dispositifs de commande et dispositifs d'interface opérateur.

Comme la classe de transport 0 ne détecte pas les doublons de paquets de données, l'application-cible a la responsabilité de les détecter. Un doublon de paquet de données est une réémission du dernier paquet de données envoyé. La Figure 34 montre les actions qui ont lieu au cours d'un transfert de données utilisant la classe de transport client 0 et la classe de transport serveur 0.



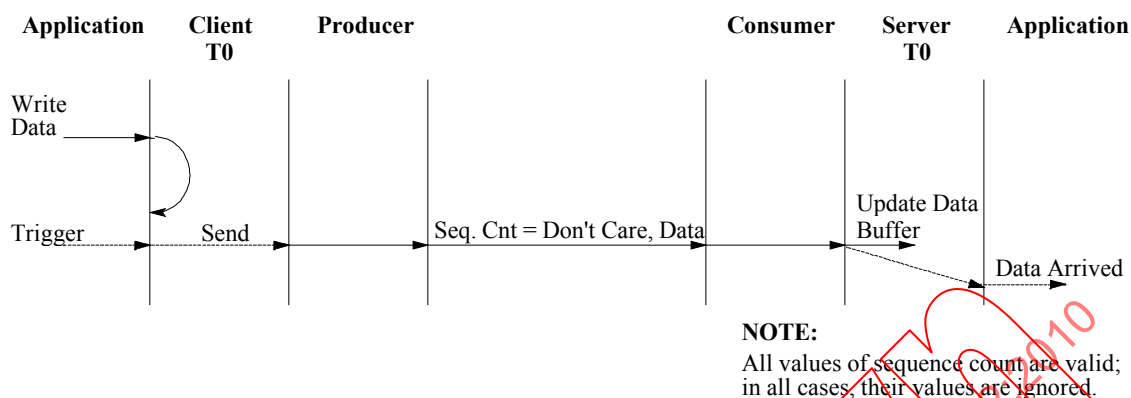
Légende

Anglais	Français
Application	Application
Client Transport Class 0	Classe de transport client 0
Trigger	Trigger
Data	Données
Send	Envoyer
T-PDU Buffer	Tampon des T-PDU
T-PDU Packet	Paquet de T-PDU
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Data Arrived	Données arrivées
Serveur Transport Class 0	Classe de transport serveur 0
Packet arrived	Paquet arrivé

NOTE L'instance de transport client n'écrit pas un compte de séquences dans le tampon des T-PDU côté client et l'instance de transport serveur ne lit pas un compte de séquences dans le tampon des T-PDU côté serveur. L'en-tête de transport est défini comme un "don't care" (c'est-à-dire: sans influence).

Figure 34 – Diagramme de flux de données utilisant une classe de transport client 0 et une classe de transport serveur 0

La Figure 35 montre la séquence dans laquelle les actions ont lieu pendant un transfert de données utilisant la classe de transport client 0 et la classe de transport serveur 0.



Légende

Anglais	Français
Application	Application
Client T0	Client T0
Producer	Producteur
Consumer	Consommateur
Server T0	Serveur T0
Application	Application
Data Arrived	Données arrivées
Don't care	Sans importance
Seq. Cnt = Don't Care, Data	Seq. Cnt = Don't Care, Data (compte de séquence = En-tête sans importance, Données)
Write Data	Écrire des données
Send	Envoyer
Trigger	Déclencher
Update Data Buffer	Mettre à jour le tampon des données
NOTE: All values of sequence count are valid; in all cases, their values are ignored.	NOTE: Toutes les valeurs du compte de séquence sont valides; dans tous les cas, leurs valeurs sont ignorées

Figure 35 – Diagramme de séquence du transfert de données utilisant la classe de transport 0

9.3.5.2 Classe de transport client 0

9.3.5.2.1 Fonctions

La fonction de classe de transport client 0 est juste de transmettre le service *trigger* (déclencher) d'une application vers le producteur comme un service *send* (envoyer). Il est utile que la classe de transport client 0 soit montrée du point de vue conception de haut niveau. Dans les mises en œuvre réelles, l'application peut déclencher directement le producteur. Un état de repos ("idle") est fourni pour la cohérence avec d'autres classes de transport. Dans l'état *idle*, tous les événements, à l'exception de *delete* et de *start*, sont ignorés.

Un transport client de classe 0 a la responsabilité:

- d'accepter le déclencheur de l'application client qu'il a écrit un paquet de données dans le tampon des TPDU côté client,
- d'écrire un en-tête de transport dans le tampon de TPDU pour chaque paquet que l'application client écrit dans le tampon de TPDU côté client (facultatif),
- de déclencher le nœud producteur de la connexion réseau pour produire le paquet TPDU sur la liaison et l'envoyer vers le nœud consommateur de la connexion réseau.

9.3.5.2.2 États

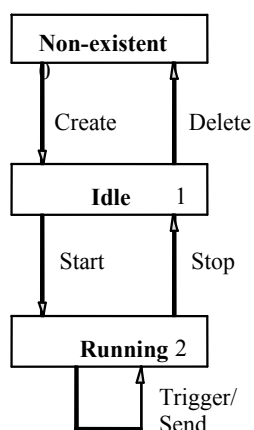
Les états définis et leurs descriptions du client de transport de classe 0 sont énumérés dans le Tableau 227.

Tableau 227 – États du client de transport de classe 0

État	Description
Non-existent (inexistant)	L'instance de classe de transport 0 n'existe pas.
Idle	L'instance de classe de transport 0 a été créée, mais n'a pas encore démarré.
Running	L'instance de classe de transport 0 a été créée et a démarré.

9.3.5.2.3 Transitions d'états

Le diagramme de transitions d'états du transport client de classe 0 est montré à la Figure 36.



Légende

Anglais	Français
Create	Créer
Delete	Supprimer
Running	En marche
Trigger/ Send	Déclencher/ Envoyer
Start	Démarrer
Stop	Arrêter
Idle	Au repos
Non-existent	Inexistant

Figure 36 – STD client de classe 0

9.3.5.2.4 Matrice d'événements d'états

La matrice d'événements d'états du transport client de classe 0 est montrée dans le Tableau 228.

Tableau 228 – SEM client de classe 0

Événement	État		
	Non-existent	Idle	Running
Create	Passer à Idle	Erreur	Erreur
Delete	Erreur	Passer à Non-existent	Erreur
Start	Erreur	Passer à Running	Erreur
Stop	Erreur	Aucune action	Passer à Idle
Trigger	Erreur	Aucune action	Send

9.3.5.3 Classe de transport serveur 0

9.3.5.3.1 Fonction

La fonction de la classe de transport serveur 0 est de transmettre l'événement *packet arrived* (paquet arrivé) du consommateur au producteur comme un événement *data arrived* (données arrivées). Il est utile que la classe de transport serveur 0 soit montrée du point de vue

conception de haut niveau. Dans des mises en œuvre réelles, l'application peut recevoir l'événement directement du consommateur. Dans l'état *idle*, toutes les arrivées de paquets sont ignorées.

Un transport serveur de classe 0 a la responsabilité:

- d'accepter la notification issue du nœud consommateur de la connexion réseau qu'il a écrit un paquet de données dans le tampon des TPDU côté serveur,
- de localiser et de lire l'en-tête de transport de chaque paquet que le nœud consommateur de la connexion réseau écrit dans le tampon des TPDU côté serveur,
- de notifier à l'application serveur que des données ont été écrites dans le tampon des TPDU côté serveur.

9.3.5.3.2 États

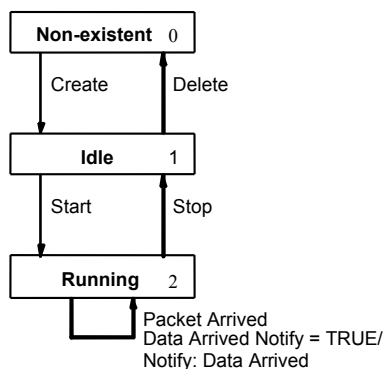
Les états définis et leurs descriptions du serveur de transport de classe 0 sont énumérés dans le Tableau 229.

Tableau 229 – États du serveur de transport de classe 0

État	Description
Non-existent	L'instance de classe de transport 0 n'existe pas.
Idle	L'instance de classe de transport 0 a été créée, mais n'a pas encore démarré.
Running	L'instance de classe de transport 0 a été créée et a démarré.

9.3.5.3.3 Transitions d'états

Les transitions d'états du transport serveur de classe 0 sont montrées à la Figure 37.



Légende

Anglais	Français
Non-existent	Inexistant
Create	Créer
Delete	Supprimer
Idle	Au repos
Start	Démarrer
Stop	Arrêter
Running	En marche
Packet Arrived	Paquet arrivé
Data Arrived Notify = TRUE/ Notify: Data Arrived	Données arrivées, notifier = vrai Notifier: Données arrivées

Figure 37 – STD serveur de classe 0

9.3.5.3.4 Matrice d'événements d'états

La matrice d'événements d'états du transport serveur de classe 0 est montrée dans le Tableau 230.

Tableau 230 – SEM serveur de classe 0

Événement	État		
	Non-existent	Idle	Running
Create	Passer à Idle	Erreur	Erreur
Delete	Erreur	Passer à Non-existent	Erreur
Start	Erreur	Passer à Running	Erreur
Stop	Erreur	Aucune action	Passer à Idle
Packet Arrived & Data Arrived "Notify" = TRUE	Aucune action	Aucune action	Notifier: Données arrivées

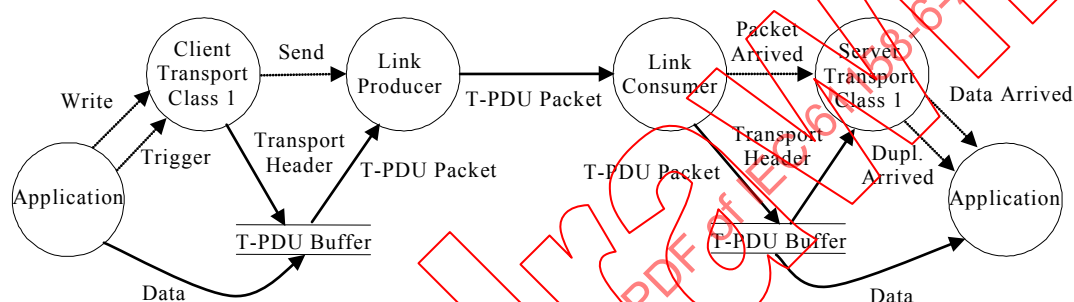
9.3.6 Diagrammes d'états de transport – classe 1

9.3.6.1 Fonctions

Comme la classe 0, la classe de transport 1 doit autoriser une connexion point à point ou multipoint. Contrairement à la classe 0, qui n'a pas d'en-tête de transport, le transport de classe 1 doit inclure un numéro de séquence, qui est utilisé pour détecter la distribution de doublons de paquets de données.

NOTE 1 La classe 1 est préférée à la classe 0 pour les cibles parce que l'application-cible n'a pas à accomplir de détection de doublons, ce qui réduit le surdébit exigé dans les nœuds d'extrémité qui prennent en charge le déclencheur de transport de changement d'état. La classe 1 permet un déclencheur de changement d'état efficace, car elle a été conçue pour autoriser les transferts de données de changement d'état.

La Figure 38 montre les actions qui ont lieu au cours d'un transfert de données utilisant une instance de classe de transport client 1 et une instance de classe de transport serveur 1.



Légende

Anglais	Français
Application	Application
Client Transport Class 1	Classe de transport client 1
Trigger	Trigger
Data	Données
Send	Envoyer
T-PDU Buffer	Tampon des T-PDU
T-PDU Packet	Paquet de T-PDU
Transport Header	En-tête de transport
Link Producer	Producteur de liaison
Link Consumer	Consommateur de liaison
Data Arrived	Données arrivées
Serveur Transport Class 1	Classe de transport serveur 1
Packet arrived	Paquet arrivé
Dupl. arrived	Doublon arrivé

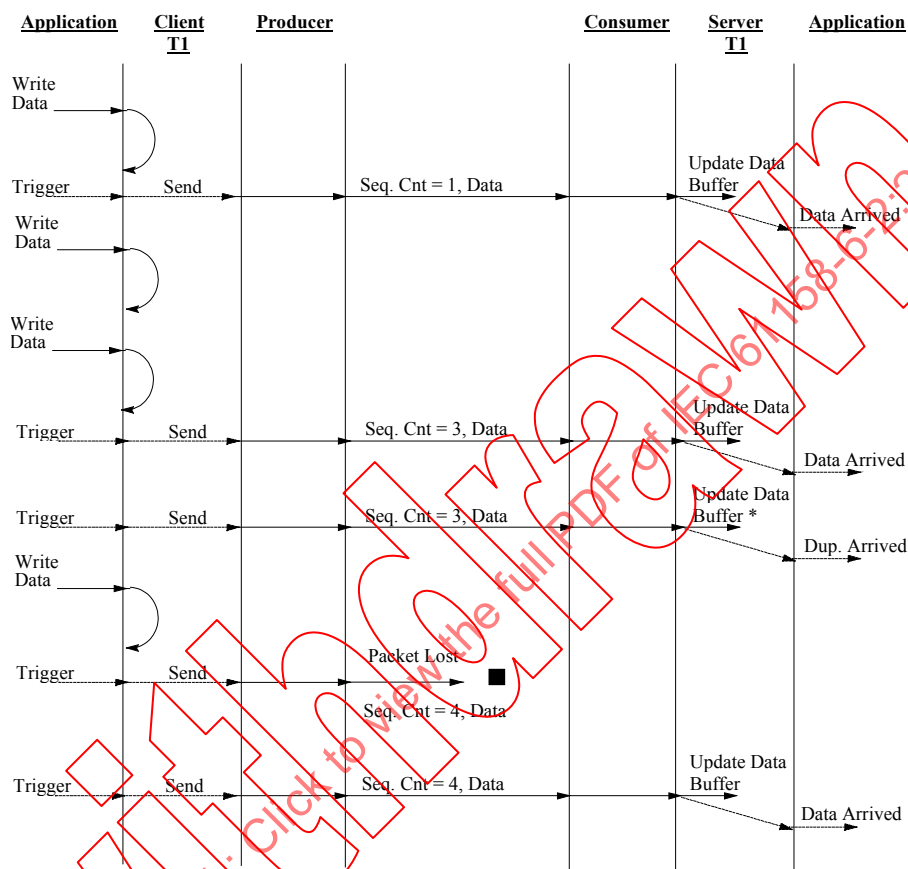
NOTE Le paquet de T-PDU comprend l'en-tête de transport issu du transport client et les données issues de l'application.

Figure 38 – Diagramme de flots de données utilisant la classe de transport client 1 et la classe de transport serveur 1

La Figure 39 montre la séquence dans laquelle les actions ont lieu pendant un transfert de données utilisant la classe de transport client 1 et la classe de transport serveur 1.

NOTE 2 Le compte de séquences est incrémenté avec chaque écriture. En outre, lorsque le paquet est perdu sur un nouvel échantillon de données et le paquet est déclenché et envoyé ultérieurement, les données reçues par le client sont traitées comme des nouvelles données. La raison en est que c'est la première fois que le serveur a reçu ce compte de séquences. Ce mécanisme fournit une tolérance aux pannes pour les échantillons qui changent rarement.

NOTE 3 C'est un choix de mise en œuvre que de décider de mettre à jour ou non le tampon de données à la suite de la réception de doublons de données, en fonction de l'impact potentiel du traitement complémentaire.



Légende

Anglais	Français
Application	Application
Client T1	T1 client
Consumer	Consommateur
Data	Données
Data Arrived	Données arrivées
Dup. Arrived	Doublons arrivés
Packet Lost	Paquet perdu
Producer	Producteur
Send	Envoyer
Seq. Cnt = 1, Data	Seq. Cnt («Cmpte séquence») = 1, Données
Seq. Cnt = 3, Data	Seq. Cnt = 3, Données
Seq. Cnt = 4, Data	Seq. Cnt = 4, Données
Server T1	T1 serveur
Trigger	Déclencher
Update Data Buffer	Mettre à jour le tampon de données
Update Data Buffer *	Mettre à jour le tampon de données*
Write	Écrire
Write Data	Écrire des données

Figure 39 – Diagramme de séquence de transfert de données utilisant la classe de transport client 1 et la classe de transport serveur 1

La classe de transport 1 peut être utilisée avec un temporisateur heartbeat pour maintenir ouverte une connexion lorsqu'il existe des périodes où aucune donnée n'est émise. Le temporisateur heartbeat doit être initialisé chaque fois que le nœud producteur de la connexion réseau émet un paquet et doit être réglé de sorte que sa valeur maximale, ou la durée maximale entre des émissions de données, soit inférieure à la valeur de temporisation du chemin pour la connexion réseau. Lorsque le temporisateur heartbeat atteint sa valeur maximale, il déclenche l'instance de transport client et, de ce fait, le nœud producteur produit et réémet le paquet dans le tampon des TPDU côté client avant que la connexion ne temporise. Le nœud consommateur de la connexion réseau, reconnaissant que le paquet a le même compte de séquences que le dernier paquet qu'il a reçu, peut ou peut ne pas écraser en écriture le paquet dans le tampon des TPDU du côté serveur; cette décision de mise en œuvre est laissée à la discrétion des développeurs de produits, car ils peuvent le mieux estimer l'impact du traitement complémentaire.

Des applications pourraient utiliser la classe de transport pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion. Pour maintenir la connexion, un temporisateur peut être utilisé pour déclencher la production des données courantes dans le tampon des TPDU. Pour émettre de nouvelles données, l'application écrirait d'abord dans le tampon des TPDU puis déclencherait le transport client.

Des applications pourraient utiliser la classe de transport pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion. Pour maintenir la connexion, les événements "données arrivées" et "doublons arrivés" peuvent soumis ensemble à une opération OU logique pour réinitialiser un temporisateur. Si ce temporisateur expirait, l'application saurait qu'aucune donnée n'avait été reçue dans le délai indiqué. L'événement "données arrivées" identifierait de nouveaux échantillons de données. Ceci peut permettre à des applications de réduire leur surdébit en ne traitant que les nouveaux échantillons de données.

Les utilisations possibles de la classe de transport 1 comprennent les entrées échantillonnées, les sorties normalisées, les transferts de blocs cycliques, les événements de diagnostic et la communication entre dispositifs de commande et dispositifs d'interface opérateur.

9.3.6.2 Classe de transport client 1

9.3.6.2.1 Fonctions

La classe de transport 1 commence par le comportement de classe 0 et ajoute un compte de séquences. Ce compte de séquences est incrémenté par la classe de transport client lorsque des données sont écrites dans le tampon de données. Lorsque le transport reçoit un événement *write*, il doit incrémenter le compte de séquences. Lorsqu'un événement *trigger* est reçu, le transport doit mettre à jour le compte de séquences dans le tampon des TPDU et invoquer le service *send*.

Des applications pourraient utiliser la classe de transport pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion. Pour maintenir la connexion, un temporisateur peut être utilisé pour déclencher la production des données courantes dans le tampon des TPDU. Pour émettre de nouvelles données, l'application écrirait d'abord dans le tampon des TPDU puis déclencherait le transport client.

Un transport client de classe 1 a la responsabilité:

- d'accepter le déclencheur de l'application client qu'il a écrit un paquet de données dans le tampon des TPDU côté client,
- d'écrire un en-tête de transport dans le tampon des TPDU pour chaque paquet que l'application client écrit dans le tampon des TPDU côté client,

- d'initialiser le compte de séquences dans l'en-tête de transport (pour le premier paquet émis) ou d'incrémenter le compte de séquences (pour les paquets ultérieurs de la même émission),
- de déclencher le nœud producteur de la connexion réseau pour produire le paquet TPDU sur la liaison et l'envoyer vers le nœud consommateur de la connexion réseau.

9.3.6.2.2 États

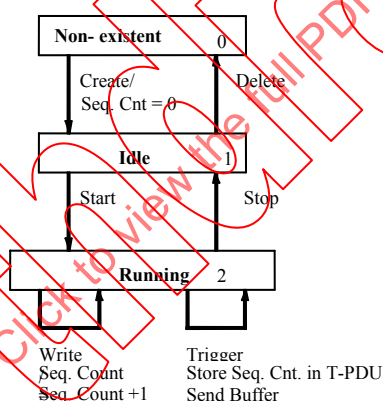
Les états définis et leurs descriptions du client de transport de classe 1 sont énumérés dans le Tableau 231.

Tableau 231 – États du client de transport de classe 1

État	Description
Non-existent	L'instance de classe de transport 1 n'existe pas.
Idle	L'instance de classe de transport 1 a été créée, mais n'a pas encore démarré.
Running	L'instance de classe de transport 1 a été créée et a démarré.

9.3.6.2.3 Transitions d'états

Les transitions d'états du transport client de classe 1 sont montrées à la Figure 40.



Légende

Anglais	Français
Non-existent	Inexistant
Create Seq. count = 0	Créer Seq. count = 0
Delete	Supprimer
Idle	Au repos
Start	Démarrer
Stop	Arrêter
Running	En marche
Write	écrire
Seq. count	Seq. count («compteur de séquence»)
Seq. count + 1	Seq. count + 1

NOTE Le compte de séquences est mis à 0 par le service *create*. Il est ensuite incrémenté après chaque événement *write* tant qu'il n'a pas atteint une valeur maximale de $2^{16}-1$. Après quoi, il repasse par 0.

Figure 40 – STD client de classe 1

9.3.6.2.4 Matrice d'événements d'états

Les transitions d'états du transport client de classe 1 sont montrées dans le Tableau 232.

Tableau 232 – SEM client de classe 1

Événement	État		
	Non-existant	Idle	Running
Create	1) Passer à Idle 2) Mettre seq. count = 0	Erreur	Erreur
Delete	Erreur	Passer à Non-existant	Erreur
Start	Erreur	Passer à Running	Erreur
Stop	Erreur	Aucune action	Passer à Idle
Write	Erreur	Aucune action	1) Seq. Cnt = Seq. Cnt + 1
Trigger	Erreur	Aucune action	1) Stocker Seq. Cnt. dans le tampon des TPDU. 2) Envoyer

9.3.6.3 Classe de transport serveur 1

9.3.6.3.1 Fonctions

La classe de transport 1 commence par le comportement de classe 0 et ajoute un compte de séquences. Le compte de séquences reçu est comparé au précédent compte de séquences. S'ils sont égaux, le paquet reçu est considéré comme étant un doublon de paquet. S'ils ne sont pas égaux, les données reçues sont considérées comme étant un nouvel échantillon. Il n'est effectué aucune tentative de compter combien de comptes de séquences ont changé entre les échantillons.

Des applications pourraient utiliser la classe de transport pour distinguer entre échantillons nouveaux et échantillons anciens qui avaient été envoyés pour maintenir la connexion. Pour maintenir la connexion, les événements "données arrivées" et "doublons arrivés" peuvent soumis ensemble à une opération OU logique pour réinitialiser un temporisateur. Si ce temporisateur expirait, l'application saurait qu'aucune donnée n'avait été reçue dans le délai indiqué. L'événement "données arrivées" identifierait de nouveaux échantillons de données. Ceci peut permettre à des applications de réduire leur surdébit en ne traitant que les nouveaux échantillons de données.

Un transport serveur de classe 1 a la responsabilité:

- d'accepter la notification issue du nœud consommateur de la connexion réseau qu'il a écrit un paquet de données dans le tampon des TPDU côté serveur,
- de localiser et de lire l'en-tête de transport de chaque paquet que le nœud consommateur de la connexion réseau écrit dans le tampon des TPDU côté serveur,
- de déclencher l'application serveur que des données ont été écrites dans le tampon des TPDU côté serveur.
- de comparer le compte de séquences de chaque paquet avec celui du paquet précédent,
- de notifier à l'application serveur que le paquet arrivé le plus récemment est un doublon du précédent lorsque leurs comptes de séquences concordent.