

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 du protocole de la couche application – Eléments
de type 2**



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2014 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 l'IEC ou du Comité national de l'IEC du pays du demandeur. Si vous avez des questions sur le copyright de l'IEC ou si vous désirez obtenir des droits supplémentaires sur cette publication, utilisez les coordonnées ci-après ou contactez le Comité national de l'IEC 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.

IEC Catalogue - webstore.iec.ch/catalogue

The stand-alone application for consulting the entire bibliographical information on IEC International Standards, Technical Specifications, Technical Reports and other documents. Available for PC, Mac OS, Android Tablets and iPad.

IEC publications search - www.iec.ch/searchpub

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 It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

Stay up to date on all new IEC publications. Just Published details all new publications released. Available online and 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 14 additional languages. Also known as the International Electrotechnical Vocabulary (IEV) online.

IEC Glossary - std.iec.ch/glossary

More than 55 000 electrotechnical terminology entries in English and French extracted from the Terms and Definitions clause of IEC publications issued since 2002. Some entries have been collected from earlier publications of IEC TC 37, 77, 86 and CISPR.

IEC Customer Service Centre - webstore.iec.ch/csc

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

A propos de l'IEC

La Commission Electrotechnique Internationale (IEC) 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 IEC

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

Catalogue IEC - webstore.iec.ch/catalogue

Application autonome pour consulter tous les renseignements bibliographiques sur les Normes internationales, Spécifications techniques, Rapports techniques et autres documents de l'IEC. Disponible pour PC, Mac OS, tablettes Android et iPad.

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

La recherche avancée permet de trouver des publications IEC 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.

IEC Just Published - webstore.iec.ch/justpublished

Restez informé sur les nouvelles publications IEC. 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 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 14 langues additionnelles. Egalement appelé Vocabulaire Electrotechnique International (IEV) en ligne.

Glossaire IEC - std.iec.ch/glossary

Plus de 55 000 entrées terminologiques électrotechniques, en anglais et en français, extraites des articles Termes et Définitions des publications IEC parues depuis 2002. Plus certaines entrées antérieures extraites des publications des CE 37, 77, 86 et CISPR de l'IEC.

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 du 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 978-2-8322-1756-6

**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.....	18
3.1 Terms and definitions from other ISO/IEC standards.....	18
3.2 Terms and definitions from IEC 61158-5-2.....	19
3.3 Additional terms and definitions.....	19
3.4 Abbreviations and symbols.....	26
3.5 Conventions.....	27
4 Abstract syntax.....	32
4.1 FAL PDU abstract syntax.....	32
4.2 Data abstract syntax specification.....	149
4.3 Encapsulation abstract syntax.....	154
5 Transfer syntax.....	171
5.1 Compact encoding.....	171
5.2 Data type reporting.....	179
6 Structure of FAL protocol state machines.....	186
7 AP-Context state machine.....	187
7.1 Overview.....	187
7.2 Connection object state machine.....	187
8 FAL service protocol machine (FSPM).....	195
8.1 General.....	195
8.2 Primitive definitions.....	195
8.3 Parameters of primitives.....	199
8.4 FSPM state machines.....	200
9 Application relationship protocol machines (ARPMs).....	200
9.1 General.....	200
9.2 Connection-less ARPM (UCMM).....	201
9.3 Connection-oriented ARPMs (transports).....	210
10 DLL mapping protocol machine 1 (DMPM 1).....	238
10.1 General.....	238
10.2 Link producer.....	238
10.3 Link consumer.....	239
10.4 Primitive definitions.....	239
10.5 DMPM state machine.....	241
10.6 Data-link Layer service selection.....	243
11 DLL mapping protocol machine 2 (DMPM 2).....	243
11.1 General.....	243
11.2 Mapping of UCMM PDUs.....	243
11.3 Mapping of transport class 0 and class 1 PDUs.....	251
11.4 Mapping of transport class 2 and class 3 PDU's.....	251

11.5 Mapping of transport classes 4 to 6	253
11.6 IGMP Usage	253
11.7 Quality of Service (QoS) for CP 2/2 messages	254
11.8 Management of an encapsulation session	258
12 DLL mapping protocol machine 3 (DMPM 3)	258
Bibliography	259

Figure 1 – Attribute table format and terms	27
Figure 2 – Service request/response parameter	28
Figure 3 – Example of an STD	31
Figure 4 – Network connection parameters	54
Figure 5 – Time tick	57
Figure 6 – Connection establishment time-out	59
Figure 7 – Member ID/EX description (WORD)	71
Figure 8 – Transport Class Trigger attribute	103
Figure 9 – CP2/3_initial_comm_characteristics attribute format	107
Figure 10 – Segment type	116
Figure 11 – Port segment	117
Figure 12 – Logical segment encoding	119
Figure 13 – Extended network segment	124
Figure 14 – Symbolic segment encoding	125
Figure 15 – Encapsulation message	154
Figure 16 – FixedLengthBitString compact encoding bit placement rules	176
Figure 17 – Example compact encoding of a SWORD FixedLengthBitString	176
Figure 18 – Example compact encoding of a WORD FixedLengthBitString	176
Figure 19 – Example compact encoding of a DWORD FixedLengthBitString	176
Figure 20 – Example compact encoding of a LWORD FixedLengthBitString	176
Figure 21 – Example 1 of formal encoding of a structure type specification	181
Figure 22 – Example 2 of formal encoding of a structure type specification	182
Figure 23 – Example 3 of formal encoding of a handle structure type specification	182
Figure 24 – Example 4 of formal encoding of a handle structure type specification	183
Figure 25 – Example 5 of abbreviated encoding of a structure type specification	183
Figure 26 – Example 1 of formal encoding of an array type specification	184
Figure 27 – Example 2 of formal encoding of an array type specification	185
Figure 28 – Example 1 of abbreviated encoding of an array type specification	186
Figure 29 – Example 2 of abbreviated encoding of an array type specification	186
Figure 30 – I/O Connection object state transition diagram	187
Figure 31 – Bridged Connection object state transition diagram	191
Figure 32 – Explicit Messaging Connection object state transition diagram	192
Figure 33 – State transition diagram of UCMM client9	203
Figure 34 – State transition diagram of high-end UCMM server	205
Figure 35 – State transition diagram of low-end UCMM server	207
Figure 36 – Sequence diagram for a UCMM with one outstanding message	208
Figure 37 – Sequence diagram for a UCMM with multiple outstanding messages	209

Figure 38 – TPDU buffer	210
Figure 39 – Data flow diagram using a client transport class 0 and server transport class 0	213
Figure 40 – Sequence diagram of data transfer using transport class 0	213
Figure 41 – Class 0 client STD	214
Figure 42 – Class 0 server STD	215
Figure 43 – Data flow diagram using client transport class 1 and server transport class 1	216
Figure 44 – Sequence diagram of data transfer using client transport class 1 and server transport class 1	217
Figure 45 – Class 1 client STD	219
Figure 46 – Class 1 server STD	220
Figure 47 – Data flow diagram using client transport class 2 and server transport class 2	222
Figure 48 – Diagram of data transfer using client transport class 2 and server transport class 2 without returned data	223
Figure 49 – Sequence diagram of data transfer using client transport class 2 and server transport class 2 with returned data	224
Figure 50 – Class 2 client STD	225
Figure 51 – Class 2 server STD	227
Figure 52 – Data flow diagram using client transport class 3 and server transport class 3	230
Figure 53 – Sequence diagram of data transfer using client transport class 3 and server transport class 3 without returned data	231
Figure 54 – Sequence diagram of data transfer using client transport class 3 and server transport class 3 with returned data	232
Figure 55 – Class 3 client STD	234
Figure 56 – Class 3 server STD	236
Figure 57 – Data flow diagram for a link producer and consumer	238
Figure 58 – State transition diagram for a link producer	242
Figure 59 – State transition diagram for a link consumer	242
Figure 60 – DS field in the IP header	255
Figure 61 – IEEE 802.1Q tagged frame	256
Table 1 – Get_Attribute_All response service rules	28
Table 2 – Example class level object/service specific response data of Get_Attribute_All	29
Table 3 – Example Get_Attribute_All data array method	29
Table 4 – Set_Attribute_All request service rules	30
Table 5 – Example Set_Attribute_All attribute ordering method	30
Table 6 – Example Set_Attribute_All data array method	30
Table 7 – State event matrix format	32
Table 8 – Example state event matrix	32
Table 9 – UCMM_PDU header format	36
Table 10 – UCMM command codes	36
Table 11 – Transport class 0 header	37

Table 12 – Transport class 1 header	37
Table 13 – Transport class 2 header	37
Table 14 – Transport class 3 header	37
Table 15 – Real-time data header – exclusive owner	38
Table 16 – Real-time data header– redundant owner	38
Table 17 – Forward_Open request format	42
Table 18 – Forward_Open_Good response format	43
Table 19 – Forward_Open_Bad response format	44
Table 20 – Large_Forward_Open request format	44
Table 21 – Large_Forward_Open_Good response format	45
Table 22 – Large_Forward_Open_Bad response format	46
Table 23 – Forward_Close request format	46
Table 24 – Forward_Close_Good response format	47
Table 25 – Forward_Close_Bad response format	47
Table 26 – Unconnected_Send request format	48
Table 27 – Unconnected_Send_Good response format	49
Table 28 – Unconnected_Send_Bad response format	49
Table 29 – Unconnected_Send request format (modified)	50
Table 30 – Unconnected_Send_Good response format (modified)	51
Table 31 – Unconnected_Send_Bad response format (modified)	51
Table 32 – Get_Connection_Data request format	52
Table 33 – Get_Connection_Data response format	52
Table 34 – Search_Connection_Data request format	53
Table 35 – Get_Connection_Owner request format	53
Table 36 – Get_Connection_Owner response format	53
Table 37 – Time-out multiplier	57
Table 38 – Time tick units	57
Table 39 – Encoded application path ordering	62
Table 40 – Transport class, trigger and Is_Server format	63
Table 41 – MR_Request_Header format	63
Table 42 – MR_Response_Header format	64
Table 43 – Structure of Get_Attribute_All_ResponsePDU body	64
Table 44 – Structure of Set_Attribute_All_RequestPDU body	65
Table 45 – Structure of Get_Attribute_List_RequestPDU body	65
Table 46 – Structure of Get_Attribute_List_ResponsePDU body	65
Table 47 – Structure of Set_Attribute_List_RequestPDU body	65
Table 48 – Structure of Set_Attribute_List_ResponsePDU body	65
Table 49 – Structure of Reset_RequestPDU body	66
Table 50 – Structure of Reset_ResponsePDU body	66
Table 51 – Structure of Start_RequestPDU body	66
Table 52 – Structure of Start_ResponsePDU body	66
Table 53 – Structure of Stop_RequestPDU body	66
Table 54 – Structure of Stop_ResponsePDU body	67

Table 55 – Structure of Create_RequestPDU body	67
Table 56 – Structure of Create_ResponsePDU body	67
Table 57 – Structure of Delete_RequestPDU body	67
Table 58 – Structure of Delete_ResponsePDU body	67
Table 59 – Structure of Get_Attribute_Single_ResponsePDU body	68
Table 60 – Structure of Set_Attribute_Single_RequestPDU body	68
Table 61 – Structure of Set_Attribute_Single_ResponsePDU body	68
Table 62 – Structure of Find_Next_Object_Instance_RequestPDU body	68
Table 63 – Structure of Find_Next_Object_Instance_ResponsePDU body	68
Table 64 – Structure of Apply_Attributes_RequestPDU body	69
Table 65 – Structure of Apply_Attributes_ResponsePDU body	69
Table 66 – Structure of Save_RequestPDU body	69
Table 67 – Structure of Save_ResponsePDU body	69
Table 68 – Structure of Restore_RequestPDU body	69
Table 69 – Structure of Restore_ResponsePDU body	70
Table 70 – Structure of Get_Member_ResponsePDU body	70
Table 71 – Structure of Set_Member_RequestPDU body	70
Table 72 – Structure of Set_Member_ResponsePDU body	70
Table 73 – Structure of Insert_Member_RequestPDU body	70
Table 74 – Structure of Insert_Member_ResponsePDU body	71
Table 75 – Structure of Remove_Member_ResponsePDU body	71
Table 76 – Common structure of _Member_RequestPDU body (basic format)	72
Table 77 – Common structure of _Member_ResponsePDU body (basic format)	72
Table 78 – Common structure of _Member_RequestPDU body (extended format)	72
Table 79 – Common structure of _Member_ResponsePDU body (extended format)	73
Table 80 – Extended Protocol ID	73
Table 81 – Structure of _Member_RequestPDU body (Multiple Sequential Members)	73
Table 82 – Structure of _Member_ResponsePDU body (Multiple Sequential Members)	74
Table 83 – Structure of _Member_RequestPDU body (International String Selection)	74
Table 84 – Structure of _Member_ResponsePDU body (International String Selection)	74
Table 85 – Structure of Group_Sync_RequestPDU body	75
Table 86 – Structure of Group_Sync_ResponsePDU body	75
Table 87 – Identity object class attributes	75
Table 88 – Identity object instance attributes	75
Table 89 – Identity object bit definitions for status instance attribute	77
Table 90 – Default values for extended device status field (bits 4 to 7) of status instance attribute	77
Table 91 – Class level object/service specific response data of Get_Attribute_All	77
Table 92 – Instance level object/service specific response data of Get_Attribute_All	78
Table 93 – Object-specific parameter for Reset	78
Table 94 – Reset service parameter values	78
Table 95 – Message Router object class attributes	79
Table 96 – Message Router object instance attributes	79

Table 97 – Class level object/service specific response data of Get_Attribute_All	79
Table 98 – Instance level object/service specific response data of Get_Attribute_All	80
Table 99 – Structure of Symbolic_Translation_RequestPDU body	80
Table 100 – Structure of Symbolic_Translation_ResponsePDU body	80
Table 101 – Object specific status for Symbolic_Translation service	80
Table 102 – Assembly object class attributes	81
Table 103 – Assembly object instance attributes	81
Table 104 – Assembly Instance ID ranges	81
Table 105 – Acknowledge Handler object class attributes	82
Table 106 – Acknowledge Handler object instance attributes	83
Table 107 – Structure of Add_AckData_Path_RequestPDU body	83
Table 108 – Structure of Remove_AckData_Path_RequestPDU body	83
Table 109 – Time Sync object class attributes	84
Table 110 – Time Sync object instance attributes	84
Table 111 – ClockIdentity encoding for different network implementations	87
Table 112 – ClockClass values	88
Table 113 – TimeAccuracy values	88
Table 114 – TimePropertyFlags bit values	89
Table 115 – TimeSource values	89
Table 116 – Types of Clock	89
Table 117 – Network protocol to PortPhysicalAddressInfo mapping	89
Table 118 – Parameter object class attributes	90
Table 119 – Parameter Class Descriptor bit values	90
Table 120 – Parameter object instance attributes	91
Table 121 – Semantics of Descriptor Instance attribute	92
Table 122 – Minimum and Maximum Value semantics	92
Table 123 – Scaling Formula attributes	93
Table 124 – Scaling links	94
Table 125 – Class level object/service specific response data of Get_Attribute_All	95
Table 126 – Instance level object/service specific response data of Get_Attribute_All (Parameter object stub)	95
Table 127 – Instance level object/service specific response data of Get_Attribute_All (full Parameter object)	96
Table 128 – Structure of Get_Enum_String_RequestPDU body	97
Table 129 – Structure of Get_Enum_String_ResponsePDU body	97
Table 130 – Enumerated strings Type versus Parameter data type	97
Table 131 – Connection Manager object class attributes	98
Table 132 – Connection Manager object instance attributes	98
Table 133 – Class level object/service specific response data of Get_Attribute_All	99
Table 134 – Instance level object/service specific response data of Get_Attribute_All	99
Table 135 – Instance level object/service specific request data of Set_Attribute_All	100
Table 136 – Connection object class attributes	101
Table 137 – Connection object instance attributes	101
Table 138 – Values assigned to the state attribute	102

Table 139 – Values assigned to the instance_type attribute	103
Table 140 – Possible values within Direction Bit	104
Table 141 – Possible values within Production Trigger Bits	104
Table 142 – Possible values within Transport Class Bits	105
Table 143 – TransportClass_Trigger attribute values summary	105
Table 144 – Transport Class 0 client behavior summary	106
Table 145 – Transport Class 1, 2 and 3 client behavior summary	106
Table 146 – Values defined for the CP2/3_produced_connection_id attribute	106
Table 147 – Values defined for the CP2/3_consumed_connection_id attribute	107
Table 148 – Values for the Initial Production Characteristics nibble	108
Table 149 – Values for the Initial Consumption Characteristics nibble	109
Table 150 – Values for the watchdog_timeout_action	112
Table 151 – Structure of Connection_Bind_RequestPDU body	114
Table 152 – Object specific status for Connection_Bind service	114
Table 153 – Structure of Producing_Application_Lookup_RequestPDU body	114
Table 154 – Structure of Producing_Application_Lookup_ResponsePDU body	114
Table 155 – Producing_Application_Lookup Service status codes	115
Table 156 – Possible port segment examples	117
Table 157 – TCP/IP link address examples	118
Table 158 – Extended Logical Type	119
Table 159 – Electronic key segment format	121
Table 160 – Logical segments examples	122
Table 161 – Network segments	122
Table 162 – Extended subtype definitions	124
Table 163 – Symbolic segment examples	125
Table 164 – Data segment	126
Table 165 – ANSI_Extended_Symbol segment	126
Table 166 – Addressing categories	129
Table 167 – Class code ID ranges	129
Table 168 – Attribute ID ranges	130
Table 169 – Service code ranges	130
Table 170 – Class codes	131
Table 171 – Reserved class attributes for all object class definitions	131
Table 172 – Common services list	132
Table 173 – Message Router object specific services list	133
Table 174 – Acknowledge Handler object specific services list	133
Table 175 – Parameter object specific services list	133
Table 176 – Services specific to Connection Manager	133
Table 177 – Services specific to Connection object	134
Table 178 – Device type numbering	134
Table 179 – Connection Manager service request error codes	136
Table 180 – General status codes	144
Table 181 – Extended status code for a general status of "Key Failure in path"	146

Table 182 – Identity object status codes	147
Table 183 – Encapsulation header	155
Table 184 – Encapsulation command codes	155
Table 185 – Encapsulation status codes	157
Table 186 – Nop request encapsulation header	158
Table 187 – RegisterSession request encapsulation header	158
Table 188 – RegisterSession request data portion	158
Table 189 – RegisterSession reply encapsulation header	159
Table 190 – RegisterSession reply data portion	159
Table 191 – UnRegisterSession request encapsulation header	160
Table 192 – ListServices request encapsulation header	160
Table 193 – ListServices reply encapsulation header	161
Table 194 – ListServices reply data portion	161
Table 195 – Communications capability flags	162
Table 196 – ListIdentity request encapsulation header	162
Table 197 – ListIdentity reply encapsulation header	163
Table 198 – ListIdentity reply data portion (successful)	163
Table 199 – CPF 2 identity item	164
Table 200 – ListInterfaces request encapsulation header	164
Table 201 – ListInterfaces reply encapsulation header	165
Table 202 – SendRRData request encapsulation header	165
Table 203 – SendRRData request data portion	166
Table 204 – SendRRData reply encapsulation header	166
Table 205 – SendUnitData request encapsulation header	167
Table 206 – SendUnitData request data portion	167
Table 207 – Common packet format	167
Table 208 – CPF item format	168
Table 209 – Item Type ID numbers	168
Table 210 – Null address item	169
Table 211 – Connected address item	169
Table 212 – Sequenced address item	169
Table 213 – Unconnected data item	169
Table 214 – Connected data item	170
Table 215 – Sockaddr info items	170
Table 216 – Usage of CPF items	171
Table 217 – BOOLEAN encoding	172
Table 218 – Example compact encoding of a BOOL value	172
Table 219 – Encoding of SignedInteger values	173
Table 220 – Example compact encoding of a SignedInteger value	173
Table 221 – UnsignedInteger values	173
Table 222 – Example compact encoding of an UnsignedInteger	173
Table 223 – FixedLengthReal values	173
Table 224 – Example compact encoding of a REAL value	174

Table 225 – Example compact encoding of a LREAL value	174
Table 226 – FixedLengthReal values	174
Table 227 – STRING value	174
Table 228 – STRING2 value	175
Table 229 – STRINGN value	175
Table 230 – SHORT_STRING value	175
Table 231 – Example compact encoding of a STRING value	175
Table 232 – Example compact encoding of STRING2 value	175
Table 233 – SHORT_STRING type	175
Table 234 – Example compact encoding of a single dimensional ARRAY	177
Table 235 – Example compact encoding of a multi-dimensional ARRAY	178
Table 236 – Example compact encoding of a STRUCTURE	178
Table 237 – Identification codes and descriptions of elementary data types	179
Table 238 – Identification codes and descriptions of constructed data types	180
Table 239 – Formal structure encoding definition	181
Table 240 – Formal structure with handles encoding definition	182
Table 241 – Abbreviated structure encoding definition	183
Table 242 – Formal array encoding definition	184
Table 243 – Abbreviated array encoding definition	185
Table 244 – I/O Connection state event matrix	188
Table 245 – Bridged Connection state event matrix	191
Table 246 – Explicit Messaging Connection state event matrix	193
Table 247 – Primitives issued by FAL user to FSPM	196
Table 248 – Primitives issued by FAL user to FSPM	196
Table 249 – Primitives issued by FSPM to FAL user	198
Table 250 – Parameters used with primitives exchanged between FAL user and FSPM	200
Table 251 – Primitives issued by FSPM to ARPM	202
Table 252 – Primitives issued by ARPM to FSPM	202
Table 253 – Parameters used with primitives exchanged between FSPM and ARPM	202
Table 254 – UCMM client states	203
Table 255 – State event matrix of UCMM client	204
Table 256 – High-end UCMM server states	205
Table 257 – State event matrix of high-end UCMM server	206
Table 258 – Low-end UCMM server states	207
Table 259 – State event matrix of low-end UCMM server	207
Table 260 – Notification	210
Table 261 – Transport classes	211
Table 262 – Primitives issued by FSPM to ARPM	211
Table 263 – Primitives issued by ARPM to FSPM	212
Table 264 – Parameters used with primitives exchanged between FSPM and ARPM	212
Table 265 – Class 0 transport client states	214
Table 266 – Class 0 client SEM	214
Table 267 – Class 0 transport server states	215

Table 268 – Class 0 server SEM	215
Table 269 – Class 1 transport client states	218
Table 270 – Class 1 client SEM	219
Table 271 – Class 1 transport server states	220
Table 272 – Class 1 server SEM	221
Table 273 – Class 2 transport client states	225
Table 274 – Class 2 client SEM	226
Table 275 – Class 2 transport server states	227
Table 276 – Class 2 server SEM	228
Table 277 – Class 3 transport client states	233
Table 278 – Class 3 client SEM	234
Table 279 – Class 3 transport server states	235
Table 280 – Class 3 server SEM	237
Table 281 – Primitives issued by ARPM to DMPM	239
Table 282 – Primitives issued by DMPM to ARPM	239
Table 283 – Parameters used with primitives exchanged between ARPM and DMPM	239
Table 284 – Primitives exchanged between data-link layer and DMPM	240
Table 285 – Parameters used with primitives exchanged between DMPM and Data-link	240
Table 286 – Selection of connection ID	241
Table 287 – Link producer states	241
Table 288 – State event matrix of link producer	242
Table 289 – Link consumer states	242
Table 290 – State event matrix of link consumer	243
Table 291 – UCMM request	243
Table 292 – UCMM reply	244
Table 293 – Network Connection ID selection	246
Table 294 – Sockaddr Info usage	247
Table 295 – Example multicast assignments	250
Table 296 – UDP data format for class 0 and class 1	251
Table 297 – Transport class 2 and class 3 connected data	252
Table 298 – Default DSCP and IEEE 802.1D mapping	256

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.

Attention is drawn to the fact that the use of the associated protocol type is restricted by its intellectual-property-right holders. In all cases, the commitment to limited release of intellectual-property-rights made by the holders of those rights permits a layer protocol type to be used with other layer protocols of the same type, or in other type combinations explicitly authorized by its intellectual-property-right holders.

NOTE Combinations of protocol types are specified in IEC 61784-1 and IEC 61784-2.

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

This third edition cancels and replaces the second edition published in 2010. This edition constitutes a technical revision.

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

- Updates of definition used by the Time Sync object;
- Addition of “member” and object specific services in 4.1.2.1, 4.1.8, 4.1.10, 8.2;
- Removal of obsolete transport classes 4 to 6 in 4.1.4.6, and 9.3.9 to 9.3.12;
- Clarification of transport header formats in 4.1.4;
- Update of CM and MR PDUs in 4.1.5 to 4.1.7;
- Updates of Identity object PDUs in 4.1.8.2;
- Updates of Assembly object PDUs in 4.1.8.4;
- Updates of Time sync object PDUs in 4.1.8.6;
- Updates of Parameter object PDUs in 4.1.8.7;
- Updates of Connection Manager object PDUs in 4.1.8.8;
- Updates of message and connection paths in 4.1.9;
- Updates of object class codes in 4.1.10 and error codes in 4.1.11;
- Updates of data types in 4.2.4 and 5.2.3;
- Updates of the encapsulation abstract syntax in 4.3;
- Updates to the DLL mapping protocol machine 2 in Clause 11;
- Miscellaneous editorial corrections.

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

FDIS	Report on voting
65C/764/FDIS	65C/774/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.

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 61158-1.

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

- as a guide for 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.

Withdrawn

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, 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.

A secondary objective is to provide migration paths from previously-existing industrial communications protocols.

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 documents, in whole or in part, are normatively referenced in this document and are indispensable for its application. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

NOTE All parts of the IEC 61158 series, as well as IEC 61784-1 and IEC 61784-2 are maintained simultaneously. Cross-references to these documents within the text therefore refer to the editions as dated in this list of normative references.

IEC 61158-1:2014, *Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*

IEC 61158-3-2:2014, *Industrial communication networks – Fieldbus specifications – Part 3-2: Data-link layer service definition – Type 2 elements*

IEC 61158-4-2:2014, *Industrial communication networks – Fieldbus specifications – Part 4-2: Data-link layer protocol specification – Type 2 elements*

IEC 61158-5-2:2014, *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 61800-7-202, *Adjustable speed electrical power drive systems – Part 7-202: Generic interface and use of profiles for power drive systems – Profile type 2 specification*

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

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*

ISO 639-2, *Codes for the representation of names of languages – Part 2: Alpha-3 code*

ISO 11898:1993¹, *Road vehicles – Interchange of digital information – Controller area network (CAN) for high-speed communication*

IEEE 802.1D-2004, *IEEE standard for local and metropolitan area networks – Media Access Control (MAC) bridges*, available at <http://www.ieee.org>

IEEE 802.1Q-2005¹, *IEEE standard for local and metropolitan area networks – Virtual bridged local area networks*, available at <http://www.ieee.org>

IEEE 802.3-2008, *IEEE Standard for Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 3: Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications*, available at <http://www.ieee.org>

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

IETF RFC 2597, *Assured Forwarding PHB Group*, available at <http://www.ietf.org>

IETF RFC 2873, *TCP Processing of the IPv4 Precedence Field*, available at <http://www.ietf.org>

IETF RFC 3140, *Per Hop Behavior Identification Codes*, available at <http://www.ietf.org>

¹ A newer edition of this standard has been published, but only the cited edition applies.

IETF RFC 3246, *An Expedited Forwarding PHB (Per-Hop Behavior)*, available at <http://www.ietf.org>

IETF RFC 3376, *Internet Group Management Protocol, Version 3*, available at <http://www.ietf.org>

IETF RFC 4594, *Configuration Guidelines for DiffServ Service Classes*, 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 8824-1

- a) object identifier
- b) type
- 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 1 to entry: The attributes of an object contain information about variable portions of an object. Typically, they provide status information or govern the operation of an object. Attributes may also affect the behavior of an object. Attributes are divided into class attributes and instance attributes.

3.3.5

behavior

indication of how an object responds to particular events

3.3.6

Best Master Clock Algorithm

BMCA

algorithm performed by each node to determine the clock that will become the master clock on a subnet and the grandmaster clock for the domain

Note 1 to entry: The algorithm primarily compares priority1, clock quality, priority2, and source identity to determine the best master among available candidates.

3.3.7

boundary clock

clock that has multiple Precision Time Protocol (PTP) ports in a domain and maintains the timescale used in the domain

Note 1 to entry: It may serve as the source of time, i.e., be a master clock, and may synchronize to another clock, i.e., be a slave clock.

[SOURCE: IEC 61588:2009, 3.1.3, modified – second sentence changed to a Note]

3.3.8

called

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

3.3.9

calling

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

3.3.10

class

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

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

3.3.11

class attribute

attribute that is shared by all objects within the same class

3.3.12

class code

unique identifier assigned to each object class

3.3.13

class specific service

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

Note 1 to entry: A class specific object is unique to the object class which defines it.

3.3.14

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

3.3.15

clock

node participating in the Precision Time Protocol (PTP) that is capable of providing a measurement of the passage of time since a defined epoch

Note 1 to entry: There are three types of clocks in IEC 61588:2009, boundary, transparent and ordinary clocks.

[SOURCE: IEC 61588:2009, 3.1.4, modified – different Note]

3.3.16

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, and Message Router object.

3.3.17**connection**

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

Note 1 to entry: Connections may be either point-to-point or multipoint.

3.3.18**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

3.3.19**connection path**

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

3.3.20**connection point**

buffer which is represented as a subinstance of an Assembly object

3.3.21**consume**

act of receiving data from a producer

3.3.22**consumer**

node or sink that is receiving data from a producer

3.3.23**consuming application**

application that consumes data

3.3.24**cyclic**

repetitive in a regular manner

3.3.25**device**

physical hardware connected to the link

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

3.3.26**device profile**

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

3.3.27**domain**

logical grouping of clocks that synchronize to each other using the protocol, but that are not necessarily synchronized to clocks in another domain

[SOURCE: IEC 61588:2009, 3.1.7]

3.3.28**end node**

producing or consuming node

3.3.29

end point

one of the communicating entities involved in a connection

3.3.30

epoch

origin of a time scale

[SOURCE: IEC 61588:2009, 3.1.9]

3.3.31

error

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

3.3.32

frame

denigrated synonym for DLPDU

3.3.33

grandmaster clock

within a domain, clock that is the ultimate source of time for clock synchronization using the PTP protocol

[SOURCE: IEC 61588:2009, 3.1.13]

3.3.34

instance

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

EXAMPLE California is an instance of the object class state.

Note 1 to entry: The terms object, instance, and object instance are used to refer to a specific instance.

3.3.35

instance attribute

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

3.3.36

instantiated

object that has been created in a device

3.3.37

interoperability

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

3.3.38

Keeper

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

3.3.39

little endian

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

Note 1 to entry: Native Type 2 data types are sent in little endian order.

3.3.40**Lpacket**

Link packet

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

Note 1 to entry: Peer data-link layers use Lpackets to send and receive service data units from higher layers in the OSI stack.

3.3.41**management information**

network accessible information that supports managing the operation of the fieldbus system, including the application layer

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

3.3.42**master clock**

in the context of a single Precision Time Protocol (PTP) communication path, clock that is the source of time to which all other clocks on that path synchronize

[SOURCE: IEC 61588:2009, 3.1.17]

3.3.43**member**

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

3.3.44**Message Router**

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

3.3.45**multipoint connection**

connection from one node to many

Note 1 to entry: Multipoint connections allow messages from a single producer to be received by many consumer nodes.

3.3.46**network**

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

3.3.47**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

3.3.48**object specific service**

service unique to the object class which defines it

3.3.49

ordinary clock

clock that has a single Precision Time Protocol (PTP) port in a domain and maintains the timescale used in the domain

Note 1 to entry: It may serve as a source of time, i.e., be a master clock, or may synchronize to another clock, i.e., be a slave clock.

[SOURCE: IEC 61588:2009, 3.1.22, modified – second sentence changed to a Note]

3.3.50

originator

client responsible for establishing a connection path to the target

3.3.51

parent clock

master clock to which a clock is synchronized

[SOURCE: IEC 61588:2009, 3.1.23]

3.3.52

point-to-point connection

connection that exists between exactly two application objects

3.3.53

Precision Time Protocol

PTP

protocol defined by IEC 61588:2009

Note 1 to entry: As an adjective, it indicates that the modified noun is specified in or interpreted in the context of IEC 61588:2009.

[SOURCE: IEC 61588:2009, 3.1.28, modified – second sentence changed to a Note]

3.3.54

produce

act of sending data to be received by a consumer

3.3.55

producer

node that is responsible for sending data

3.3.56

PTP message

one of the message types defined in IEC 61588:2009

[SOURCE: IEC 61588:2009, 3.1.33]

3.3.57

PTP port

logical access point of a clock for PTP communications to the communications network

[SOURCE: IEC 61588:2009, 3.1.35]

3.3.58**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

3.3.59**resource**

processing or information capability of a subsystem

3.3.60**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

3.3.61**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

3.3.62**service**

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

3.3.63**synchronized clocks**

(to a specified uncertainty) two clocks which have the same epoch and for which measurements of the time of a single event at an arbitrary time differ by no more than the specified uncertainty

[SOURCE: IEC 61588:2009, 3.1.40, modified – reworded]

3.3.64**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 1 to entry. 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.

3.3.65**target**

end-node to which a connection is established

3.3.66**temporary node**

transient node

3.3.67**transaction id**

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

3.3.68**transparent clock**

device that measures the time taken for a Precision Time Protocol (PTP) event message to transit the device and provides this information to clocks receiving this PTP event message

[SOURCE: IEC 61588:2009, 3.1.46, modified – truncated]

3.3.69

Unconnected Message Manager

UCMM

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

3.3.70

unconnected service

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

3.3.71

vendor ID

identification of each product manufacturer/vendor by a unique number

Note 1 to entry: 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 (see 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	Terminal 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. 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. The service specification defines the services that are provided by the ASE.

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

Bold font is used in this standard to highlight parameter names or important requirement elements from surrounding text. Italic font is also used in Tables and Notes for that purpose for better visibility.

3.5.2 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.

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.

3.5.3 Common services

3.5.3.1 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.”

3.5.3.2 Get_Attribute_All response

3.5.3.2.1 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. Optional attributes at the end of the list that are not implemented may be omitted in the response data. Optional attributes in the middle of the list that are not implemented shall be included in the response data, and set to specified default values. 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 for objects specified in this standard 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)							

3.5.3.2.2 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.

3.5.3.3 Set_Attribute_All request

3.5.3.3.1 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	Default values shall be specified for all optional attributes that are not implemented. If an implementation does not support an optional attribute, it shall accept the specified default value for the unsupported attribute.
3	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.
4	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.
5	The Set_Attribute_All request response for objects specified in this standard 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)							

3.5.3.3.2 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 0x15 (too much data).

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.

3.5.4 State machine conventions

3.5.4.1 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.

3.5.4.2 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.

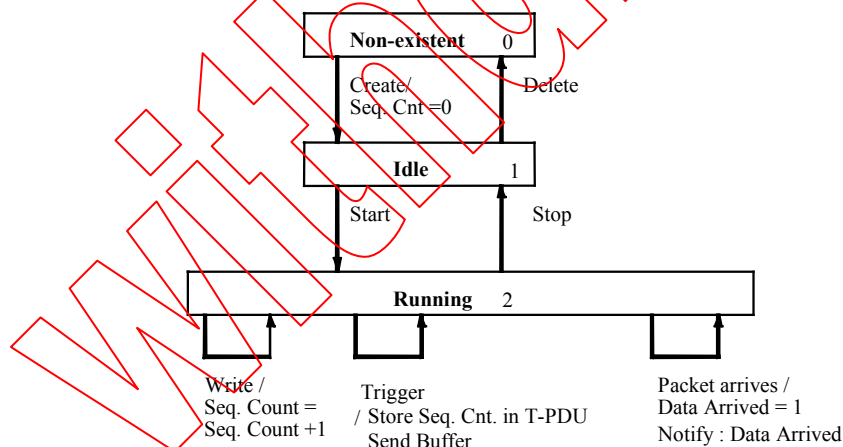


Figure 3 – Example of an STD

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

3.5.4.3 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.			

3.5.4.4 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
- Get_Member_RequestPDU
- Set_Member_RequestPDU
- Insert_Member_RequestPDU
- Remove_Member_RequestPDU
- Group_Sync_RequestPDU
- Add_AckData_Path_RequestPDU
- Remove_AckData_Path_RequestPDU
- Get_Enum_String_RequestPDU
- Symbolic_Translation_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
- Get_Member_ResponsePDU
- Set_Member_ResponsePDU
- Insert_Member_ResponsePDU
- Remove_Member_ResponsePDU
- Group_Sync_ResponsePDU
- Add_AckData_Path_ResponsePDU
- Remove_AckData_Path_ResponsePDU
- Get_Enum_String_ResponsePDU
- Symbolic_Translation_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_Connection_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_Connection_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

Subclause 4.1.3 defines the requirements for UCMM PDU's when using a Type 2 data-link layer.

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
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 to 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 $(8 | \text{timeout} \& 7) \ll (\text{timeout} \gg 3 \& 31)$.

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, 6.10 and 8.1), all UCMM 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)

NOTE Subclause contents from the previous edition have been obsoleted.

4.1.4.7 Listen only O⇒T data format

The O⇒T connection shall use a specific heartbeat format (no 32-bit header, 0 octets of application data).

4.1.4.8 Input only O⇒T data format

The O⇒T connection shall use a specific heartbeat format (no 32-bit header, 0 octets of application data).

4.1.4.9 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 15.

Table 15 – 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 to 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.

4.1.4.10 Redundant owner O⇒T data format

4.1.4.10.1 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 16.

Table 16 – Real-time data header– redundant owner

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

4.1.4.10.2 Run_idle flag

The `run_idle` flag (bit 0) shall be 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 to 31) shall be reserved and set to 0.

4.1.4.10.3 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.

4.1.4.10.4 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.

4.1.4.10.5 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.

4.1.4.10.6 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 Large_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_Connection_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. 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 Forward_Open service is used to establish a connection with a target device. This service results in local connection establishment on each link along the path.

NOTE The processing of a single Forward_Open service can result in the creation of multiple active Connection object instances.

A Forward Open can be either a non-null Forward_Open or a null Forward_Open. A non-null Forward_Open is a Forward_Open service request for which at least one of the connection types in the O2T_ or T2O_connection_parameters field is not 00 (NULL). A null Forward_Open is a Forward_Open service request for which the connection type in both the O2T_ and T2O_connection_parameters fields is 00 (NULL) and results in no connection being established.

A Forward_Open (both null and non-null) can be either not matching or matching. A matching Forward_Open service request received by the target device is one where the Connection Triad matches an existing connection.

The usage of each of the combinations of non-null/null and not matching/matching Forward_Open is described in 4.1.5.3.2 and summarized in the list below:

- non-null / not matching – open a connection;
- non-null / matching – error;
- null / not matching – ping a device, or configure;
- null / matching – reconfigure.

The response to the Forward_Open_Request_PDU is a Forward_Open_ResponsePDU. The Forward_Open response shall contain the originator 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.

4.1.5.3.2 Forward_Open variants

4.1.5.3.2.1 Non null Forward_Open, not matching

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 O2T and T2O designations; O2T means originator to target, and T2O means target to originator.

The path specified in the MR_Request_Header for the Forward_Open service shall not contain any Electronic Key segment. Electronic key segments may be included in the path specified in the connection_path parameter of the Forward_Open service.

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 interval 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.

4.1.5.3.2.2 Non null Forward_Open, matching

A non-null Forward_Open for which the Connection Triad matches an existing connection shall return a general status = 0x01, extended status = 0x0100 (Connection in use or Duplicate Forward_Open).

The suggested originator behaviour in the non-null Forward_Open, matching 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 s (first data time-out) to time-out before the connection can be re-established.

4.1.5.3.2.3 Null Forward_Open, not matching

A null Forward_Open for which the Connection Triad does not match an existing connection's parameters can be used for the following functions.

- a) Ping a device, with the following characteristics:
- the single application path "20 01 24 01" (i.e. Identity object);
 - an electronic key segment may be included;
 - no data segment is included;

- no connection is established.

b) Configure a device's application, with the following characteristics:

- a configuration application path and data segment shall be included in the request (the data is sent to the application specified by the path and applied);
- if the entire configuration cannot be applied, then none of the configuration shall be applied and the appropriate error code returned;
- an electronic key segment may be included;
- no connection is established.

A target device shall respond to a null Forward_Open request with one of the following errors when the target does not support the requested function ("ping a device" or "configure a device's application"):

- Null Forward_Open function not supported (general status 0x01, extended status 0x0132) (recommended).
- Invalid network connection parameter (general status 0x01, extended status 0x0108) (deprecated).

4.1.5.3.2.4 Null Forward_Open, matching

A Null Forward_Open for which the Connection Triad matches an existing connection's parameters can be used to reconfigure a target device's application. Intermediate nodes shall always forward these requests. The associated behaviours are:

- a configuration application path and data segment shall be included in the request and they are sent to the application to change the application configuration;
- if the entire configuration cannot be applied then none of the configuration shall be applied and the appropriate error code returned;
- the connection shall not be interrupted due to this request.

A target device shall respond to a null Forward_Open request with one of the following errors when the target does not support the requested function ("reconfigure a target device's application"):

- Null Forward_Open function not supported (general status 0x01, extended status 0x0132) (recommended).
- Invalid network connection parameter (general status 0x01, extended status 0x0108) (deprecated).

If the interpretation of the consumed/produced data changes as a result of the reconfigure operation, care shall be taken in the producing and consuming applications. There is no specified mechanism to coordinate between the producing and consuming applications, so a change in the meaning of the real time data can result in unexpected operation. Devices have the option to reject a reconfiguration request, with an Object state conflict error (general status = 0x0c), to prevent this situation.

4.1.5.3.3 Format of Forward_Open request

The format of the Forward_Open request TPDU shall be of the form shown in Table 17.

Table 17 – Forward_Open request format

Parameter name	Format
priority_and_tick	SWORD
connection time-out ticks	USINT
O2T_CID	UDINT

Parameter name	Format
T2O_CID	UDINT
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
connection timeout multiplier	USINT
reserved[3]	USINT
O2T_CM_RPI	UDINT
O2T_connection_parameters ^a	UINT
T2O_CM_RPI	UDINT
T2O_connection_parameters ^a	UINT
xport_type_and_trigger	SWORD
connection_path_size	USINT
connection_path	Padded EPATH
^a Connection type in this parameter shall be 00 (NULL) for a null Forward_Open.	

4.1.5.3.4 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 18.

Table 18 – Forward_Open_Good response format

Parameter name	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
originator_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

If a Connection object is instantiated to support this connection:

- the values of `O2T_CM_API` and `T2O_CM_API` are used for the Connection object `Expected_packet_rate` attribute, after converting to milliseconds.
- the `Expected_packet_rate` will not be used for the Connection object `Inactivity/Watchdog Timer`. See 4.1.6.2 for connection timeout handling.

4.1.5.3.5 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 19.

Table 19 – Forward_Open_Bad response format

Parameter name	Format
connection_serial_number	UINT
originator_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 `originator_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.

4.1.5.4 Large_Forward_Open

4.1.5.4.1 General

The Large_Forward_Open shall have the same function and same behaviour 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. All other requirements are identical to those specified for the Forward_Open service in 4.1.5.3.

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 20.

Table 20 – Large_Forward_Open request format

Parameter name	Format
priority_and_tick	SWORD
connection time-out ticks	USINT
O2T_CID	UDINT
T2O_CID	UDINT

Parameter name	Format
connection_serial_number	UINT
originator_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 21.

Table 21 – Large_Forward_Open_Good response format

Parameter name	Format
O2T_CID	UDINT
T2O_CID	UDINT
connection_serial_number	UINT
originator_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

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 22.

Table 22 – Large_Forward_Open_Bad response format

Parameter name	Format
connection_serial_number	UINT
originator_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.

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 23.

The path specified in the MR_Request_Header for the Forward_Close service shall not contain any Electronic Key segment. Electronic key segments may be included in the path specified in the connection_path parameter of the Forward_Close service.

Table 23 – Forward_Close request format

Parameter name	Format
connection_priority/tick time	SWORD
connection_timeout	USINT
connection_serial_number	UINT
originator_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 24.

Table 24 – Forward_Close_Good response format

Parameter name	Format
connection serial number	UINT
originator_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`, `originator_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 25.

Table 25 – Forward_Close_Bad response format

Parameter name	Format
connection_serial_number	UINT
originator_vendor_ID	UINT
originator_serial_number	UDINT
remaining_path_size	USINT
reserved	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 General

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 embedded message request shall be formatted as an OM_Request and sent to the port and link address of the last port segment using the UCMM for that link type.

NOTE The target node never sees the Unconnected_Send service but only the embedded message request arriving via the UCMM.

4.1.5.6.2 Format of Unconnected_Send request

The format of the Unconnected_Send request shall be of the form shown in Table 26.

Table 26 – Unconnected_Send request format

Parameter name	Format	Description
priority/tick time	SWORD	Used to calculate request timeout information
conn time-out ticks	USINT	
Message_size	UINT	Number of octets in the embedded Message Router PDU
Message Router PDU	OM_Request	Embedded Messenger Router PDU
pad	USINT	Only present if Message_size is an odd value. This allows the remaining fields to be aligned on a 16-bit boundary.
route_path_size	USINT	Number of 16-bit words in the route_path field
Reserved	USINT	Reserved, shall be set to zero (0)
route_path	Padded EPATH	Indicates the route to the remote target device

The path specified in the MR_Request_Header for the Unconnected_Send service shall not contain any Electronic Key segment. Electronic key segments may be included in the path specified in the MR_Request_Header of the embedded message request, and in the route_path parameter of the Unconnected_Send service.

The route path shall be of the form:

```

[Electronic Key segment1]      } Port Segment Group for 1st intermediate node
Port segment1                  }

[[Electronic Key segment2]     } Port Segment Group for 2nd intermediate node
Port segment2]                 }

...

[[Electronic Key segmentN]     } Port Segment Group for Nth intermediate node
Port segmentN]                 }
    
```

All segments listed in brackets [...] are optional. If the Electronic Key segment is present the target shall evaluate the key. See 4.1.9.3 for the definition of Port segment and 4.1.9.4.2 for the definition of Electronic Key segment.