

Green Book Edition 11

## TECHNICAL REPORT

# DLMS/COSEM Architecture and Protocols

DLMS UA 1000-2 Ed.11 V1.0  
21 December 2021

## DLMS User Association

© Copyright 1997-2021 DLMS User Association



## CONTENTS

Foreword .....	13
List of main technical changes in Edition 9 .....	15
List of main technical changes in Edition 10 .....	15
List of main technical changes in Edition 11 .....	16
1 Scope .....	17
2 Normative references .....	19
3 Terms, definitions and abbreviations and symbols .....	22
3.1 General DLMS/COSEM definitions.....	22
3.2 Definitions related to cryptographic security .....	26
3.3 Definitions and abbreviations related to the Galois/Counter Mode.....	36
3.4 Definitions and abbreviations related to Wi-SUN .....	37
3.5 General abbreviations .....	38
3.6 Symbols related to the Galois/Counter Mode.....	43
3.7 Symbols related the ECDSA algorithm .....	43
3.8 Symbols related to the key agreement algorithms .....	43
3.9 Abbreviations related to the DLMS/COSEM M-Bus communication profile .....	44
4 Information exchange in DLMS/COSEM .....	45
4.1 General .....	45
4.2 Communication model.....	45
4.3 Naming and addressing.....	46
4.4 Connection oriented operation.....	49
4.5 Application associations .....	49
4.6 Messaging patterns.....	51
4.7 Data exchange between third parties and DLMS servers .....	52
4.8 Communication profiles .....	52
4.9 Model of a DLMS/COSEM system.....	54
4.10 Model of DLMS servers .....	54
4.11 Model of a DLMS client .....	57
4.12 Interoperability and interconnectivity in DLMS/COSEM .....	58
4.13 Ensuring interconnectivity: the protocol identification service .....	58
4.14 System integration and installation .....	58
5 Physical layer services and procedures for connection-oriented asynchronous data exchange .....	59
5.1 Overview .....	59
5.2 Service specification .....	60
5.3 Protocol specification .....	64
5.4 Example: PhL service primitives and Hayes commands .....	69
6 Direct Local Connection.....	74
6.1 Introduction .....	74
6.2 METERING HDLC protocol using protocol mode E for direct local data exchange	74
6.3 Overview .....	74
6.4 Readout mode and programming mode.....	75
6.5 Physical layer – Introduction.....	76
6.6 Physical layer primitives .....	77

6.7	Data link layer.....	77
7	DLMS/COSEM transport layer for IP networks.....	78
7.1	Scope.....	78
7.2	<b>The TCP-UDP/IP based transport layers.....</b>	78
7.3	<b>The DLMS/COSEM CoAP based transport layer.....</b>	103
8	Data Link Layer using the HDLC protocol.....	151
8.1	Overview .....	151
8.2	Service specification .....	153
8.3	Protocol specification for the LLC sublayer .....	165
8.4	Protocol specification for the MAC sublayer .....	167
8.5	FCS calculation.....	187
8.6	Data link layer management services.....	191
9	DLMS/COSEM application layer.....	194
9.1	DLMS/COSEM application layer main features.....	194
9.2	Information security in DLMS/COSEM.....	204
9.3	DLMS/COSEM application layer service specification.....	262
9.4	DLMS/COSEM application layer protocol specification .....	304
9.5	Abstract syntax of COSEM PDUs .....	370
9.6	COSEM PDU XML schema.....	384
10	Using the DLMS/COSEM application layer in various communications profiles .....	407
10.1	Communication profile specific elements .....	407
10.2	The 3-layer, connection-oriented, HDLC based communication profile .....	408
10.3	The TCP-UDP/IP based communication profiles (COSEM_on_IP).....	415
10.4	<b>The CoAP based communication profile (DLMS/COSEM_on_CoAP) .....</b>	422
10.5	The S-FSK PLC profile .....	429
10.6	The wired and wireless M-Bus profile.....	456
10.7	SMS short wrapper .....	480
10.8	LPWAN profile .....	481
10.9	Wi-SUN profile .....	485
10.10	Gateway protocol .....	493
11	AARQ and AARE encoding examples.....	497
11.1	General .....	497
11.2	Encoding of the xDLMS InitiateRequest / InitiateResponse APDU .....	497
11.3	Specification of the AARQ and AARE APDU .....	500
11.4	Data for the examples .....	501
11.5	Encoding of the AARQ APDU .....	502
11.6	Encoding of the AARE APDU.....	505
12	Encoding examples: AARQ and AARE APDUs using a ciphered application context .....	511
12.1	A-XDR encoding of the xDLMS InitiateRequest APDU, carrying a dedicated key.....	511
12.2	Authenticated encryption of the xDLMS InitiateRequest APDU.....	512
12.3	The AARQ APDU .....	512
12.4	A-XDR encoding of the xDLMS InitiateResponse APDU .....	514
12.5	Authenticated encryption of the xDLMS InitiateResponse APDU .....	515
12.6	The AARE APDU .....	515
12.7	The RLRQ APDU (carrying a ciphered xDLMS InitiateRequest APDU) .....	517
12.8	The RLRE APDU (carrying a ciphered xDLMS InitiateResponse APDU) .....	517
13	S-FSK PLC encoding examples .....	519

2/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
-------	------------	-----------------------	-----------------------

13.1	CI-PDUs, ACSE APDUs and xDLMS APDUs carried by MAC frames using the IEC 61334-4-32 LLC sublayer.....	519
13.2	CI-PDUs, ACSE APDUs and xDLMS APDUs carried by MAC frames using the HDLC based LLC sublayer .....	524
13.3	Clear Alarm examples .....	529
14	Data transfer service examples.....	531
14.1	GET / Read, SET / Write examples.....	531
14.2	ACCESS service example .....	548
14.3	Compact array encoding example.....	549
14.4	Profile generic IC buffer attribute encoding examples .....	555
15	Data transfer service examples over LPWAN using LoRaWAN Technology .....	569
15.1	Example of DLMS/COSEM GET service transported through LPWAN using LoRaWAN technology .....	569
15.2	Example of DLMS/COSEM DataNotification service transported through LPWAN with SCHC Fragments .....	572
15.3	Example of DLMS/COSEM ACCES service transported through LPWAN with SCHC Fragments.....	576
	Annex A (normative) NSA Suite B elliptic curves and domain parameters .....	581
	Annex B (informative) Example of an Root-CA and end-entity Certificate using P-256 signed with P-256.....	583
B.1	Fields of public key certificates.....	583
B.2	Example of a Root-CA Certificate using P-256 signed with P-256 .....	584
B.3	Example of an end entity digital signature Certificate using P-256 signed with P-256.....	585
	Annex C (normative) Use of key agreement schemes in DLMS/COSEM .....	586
C.1	Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme.....	586
C.2	One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme .....	589
C.3	Static Unified Model C(0e, 2s, ECC CDH) scheme .....	593
	Annex D (informative) Exchanging protected xDLMS APDUs between TP and server .....	596
D.1	General .....	596
D.2	Example 1: Protection is the same in the two directions .....	596
D.3	Example 2: Protection is different in the two directions .....	598
	Bibliography .....	600
	Index .....	605

Figure 1 – The three steps approach of COSEM: Modelling – Messaging – Transporting .....	17
Figure 2 – Client–server model and communication protocols .....	46
Figure 3 – Naming and addressing in DLMS/COSEM .....	47
Figure 4 – A complete communication session in the CO environment .....	49
Figure 5 – DLMS/COSEM messaging patterns.....	51
Figure 6 – DLMS/COSEM generic communication profile .....	53
Figure 7 – Model of a DLMS/COSEM system.....	54
Figure 8 – DLMS server model.....	55
Figure 9 – Model of a DLMS client using multiple protocol stacks .....	57
Figure 10 – Typical PSTN configuration .....	59
Figure 11 – The location of the physical layer.....	60

Figure 12 – Protocol layer services of the COSEM 3-layer connection-oriented profile .....	61
Figure 13 – MSC for physical connection establishment.....	65
Figure 14 – MSC for IDENTIFY.request / .response message exchange .....	67
Figure 15 – Handling the Identification service at the server side .....	67
Figure 16 – Partial state machine for the client side physical layer .....	68
Figure 17 – MSC for physical connection request .....	70
Figure 18 – Physical connection establishment at the CALLING station .....	71
Figure 19 – MSC for physical connection establishment.....	72
Figure 20 – Data exchange between the calling and called stations .....	72
Figure 21 – MSC for a physical disconnection .....	73
Figure 22 – Entering protocol mode E (HDLC) .....	74
Figure 23 – Flow chart and switchover to METERING HDLC in protocol mode E .....	75
Figure 24 – Physical layer primitives .....	76
Figure 25 – Physical layer primitives, simplified example with one mode change only .....	76
Figure 26 – DLMS/COSEM as a standard Internet application protocol.....	79
Figure 27 – Transport layers of the DLMS/COSEM_on_IP profile .....	80
Figure 28 – Services of the DLMS/COSEM connection-less, UDP-based transport layer .....	81
Figure 29 – The wrapper protocol data unit (WPDU) .....	84
Figure 30 – The DLMS/COSEM connection-less, UDP-based transport layer PDU (UDP-PDU)	84
Figure 31 – Services of the DLMS/COSEM connection-oriented, TCP-based transport layer	86
Figure 32 – The TCP packet format.....	94
Figure 33 – TCP connection establishment.....	95
Figure 34 – TCP disconnection .....	96
Figure 35 – Data transfer using the COSEM TCP-based transport layer .....	97
Figure 36 – High-level state transition diagram for the wrapper sublayer .....	98
Figure 37 – TCP connection state diagram .....	99
Figure 38 – MSC and state transitions for establishing a transport layer and TCP connection	99
Figure 39 – MSC and state transitions for closing a transport layer and TCP connection .....	100
Figure 40 – Polling the TCP sublayer for TCP abort indication .....	101
Figure 41 – Sending an APDU in three TCP packets.....	102
Figure 42 – Receiving the message in several packets .....	103
Figure 43 – DLMS/COSEM CoAP transport protocol layer.....	104
Figure 44 – Structure of DLMS/COSEM CoAP Transport Layer .....	105
Figure 45 – CoAP client and server endpoints within the DLMS/COSEM CoAP TL .....	106
Figure 46 – Services of the connection-less DLMS/COSEM CoAP transport layer.....	109
Figure 47 – The DLMS/COSEM CoAP TL Protocol Stack .....	114
Figure 48 – The DLMS/COSEM CoAP Wrapper Protocol Data Unit (CWPDU).....	116
Figure 49 – High-level state transition diagram for the CoAP wrapper layer .....	125
Figure 50 – CoAP-DATA.request invocation handling .....	128
Figure 51 – Handling of incoming CWPDU or CoAP layer transmission failure .....	129
Figure 52 – Confirmed DLMS/COSEM AL service request through CoAP TL.....	130
Figure 53 – Piggybacked and separate response handling with reliable CoAP TL .....	132

Figure 54 – Loss Recovery of the reliable DLMS/COSEM CoAP TL .....	133
Figure 55 – Unconfirmed DataNotification through reliable CoAP TL with DLMS/COSEM CoAP TL confirmation .....	134
Figure 56 – Unconfirmed DataNotification through unreliable CoAP TL .....	135
Figure 57 – CoAP BT of a response APDU over reliable CoAP TL .....	136
Figure 58 – CoAP BT of a request APDU over reliable CoAP TL .....	137
Figure 59 – CoAP BT of request and response APDUs over reliable CoAP TL .....	138
Figure 60 – CoAP BT of an unconfirmed DataNotification over reliable CoAP TL .....	139
Figure 61 – CoAP BT of an unconfirmed DataNotification over unreliable CoAP TL .....	140
Figure 62 – CoAP BT in combination DLMS GBT for transfer of a large response APDU .....	142
Figure 63 – SET service with GBT streaming over unreliable CoAP TL .....	144
Figure 64 – SET service with GBT streaming and loss recovery by reliable CoAP TL .....	145
Figure 65 – Confirmed GET service with GBT streaming over unreliable CoAP TL .....	147
Figure 66 – Confirmed GET service with GBT streaming over reliable CoAP TL .....	148
Figure 67 – Confirmed service request with GBT streaming in both directions over unreliable CoAP TL .....	149
Figure 68 – Data link layer services for data link connection .....	154
Figure 69 – Data link layer services for data link disconnection .....	158
Figure 70 – Data link layer data transfer services .....	162
Figure 71 – Physical layer services used by the MAC sublayer .....	165
Figure 72 – The ISO/IEC 8802-2 LLC PDU format .....	165
Figure 73 – LLC format as used in DLMS/COSEM .....	165
Figure 74 – MAC sublayer frame format (HDLC frame format type 3) .....	167
Figure 75 – Multiple frames .....	167
Figure 76 – The frame format field .....	167
Figure 77 – Valid server address structures .....	169
Figure 78 – Address example .....	170
Figure 79 – MSC for long MSDU transfer in a transparent manner .....	181
Figure 80 – Example configuration to illustrate broadcasting .....	182
Figure 81 – Sending out a pending UI frame with a .response data .....	183
Figure 82 – Sending out a pending UI frame with a response to a RR frame .....	184
Figure 83 – Sending out a pending UI frame on receipt of an empty UI frame .....	184
Figure 84 – State transition diagram for the server MAC sublayer .....	187
Figure 85 – Layer management services .....	191
Figure 86 – The structure of the DLMS/COSEM application layers .....	194
Figure 87 – The concept of composable xDLMS messages .....	200
Figure 88 – Summary of DLMS/COSEM AL services .....	203
Figure 89 – Authentication mechanisms .....	205
Figure 90 – Client – server message security concept .....	208
Figure 91 – End-to-end message security concept .....	209
Figure 92 – Hash function .....	211
Figure 93 – Encryption and decryption .....	212
Figure 94 – Message Authentication Codes (MACs) .....	213

Figure 95 – GCM functions .....	214
Figure 96 – Digital signatures .....	220
Figure 97 – C(2e, 0s) scheme: each party contributes only an ephemeral key pair.....	221
Figure 98 – C(1e, 1s) schemes: party U contributes an ephemeral key pair, and party V contributes a static key pair .....	222
Figure 99 – C(0e, 2s) scheme: each party contributes only a static key pair .....	224
Figure 100 – Architecture of a Public Key Infrastructure (example).....	234
Figure 101 – MSC for provisioning the server with CA certificates .....	243
Figure 102 – MSC for security personalisation of the server.....	244
Figure 103 – Provisioning the server with the certificate of the client.....	245
Figure 104 – Provisioning the client / third party with a certificate of the server .....	246
Figure 105 – Remove certificate from the server.....	246
Figure 106 – Cryptographic protection of information using AES-GCM .....	249
Figure 107 – Structure of service-specific global / dedicated ciphering xDLMS APDUs .....	251
Figure 108 – Structure of general-glo-ciphering and general-ded-ciphering xDLMS APDUs.	252
Figure 109 – Structure of general-ciphering xDLMS APDUs .....	253
Figure 110 – Structure of general-signing APDUs .....	258
Figure 111 – Service primitives .....	262
Figure 112 – Time sequence diagrams.....	263
Figure 113 – Additional service parameters to control cryptographic protection and GBT ....	273
Figure 114 – Partial state machine for the client side control function.....	305
Figure 115 – Partial state machine for the server side control function .....	306
Figure 116 – MSC for successful AA establishment preceded by a successful lower layer connection establishment .....	314
Figure 117 – Graceful AA release using the A-RELEASE service .....	319
Figure 118 – Graceful AA release by disconnecting the supporting protocol layer .....	320
Figure 119 – Aborting an AA following a PH-ABORT.indication .....	321
Figure 120 – MSC of the GET service .....	324
Figure 121 – MSC of the GET service with block transfer.....	325
Figure 122 – MSC of the GET service with block transfer, long GET aborted .....	327
Figure 123 – MSC of the SET service.....	328
Figure 124 – MSC of the SET service with block transfer .....	328
Figure 125 – MSC of the ACTION service .....	330
Figure 126 – MSC of the ACTION service with block transfer .....	331
Figure 127 – ACCESS service with long response .....	332
Figure 128 – ACCESS service with long request and response .....	332
Figure 129 – MSC for the DataNotification service, case 1).....	333
Figure 130 – MSC for the DataNotification service, case 2).....	334
Figure 131 – MSC for the DataNotification service, case 3).....	335
Figure 132 – MSC of the Read service used for reading an attribute .....	339
Figure 133 – MSC of the Read service used for invoking a method .....	339
Figure 134 – MSC of the Read service used for reading an attribute, with block transfer .....	340
Figure 135 – MSC of the Write service used for writing an attribute .....	343

Figure 136 – MSC of the Write service used for invoking a method .....	343
Figure 137 – MSC of the Write service used for writing an attribute, with block transfer .....	344
Figure 138 – MSC of the UnconfirmedWrite service used for writing an attribute.....	345
Figure 139 – Partial service invocations and GBT APDUs .....	347
Figure 140 – The GBT procedure .....	350
Figure 141 – Send GBT APDU stream sub-procedure.....	354
Figure 142 – Process GBT APDU sub-procedure.....	356
Figure 143 – Check RQ and fill gaps sub-procedure .....	358
Figure 144 – GET service with GBT, switching to streaming.....	359
Figure 145 – GET service with partial invocations, GBT and streaming, recovery of 4 <sup>th</sup> block sent in the 2nd stream .....	360
Figure 146 – GET service with partial invocations, GBT and streaming, recovery of 4 <sup>th</sup> and 5 <sup>th</sup> block.....	361
Figure 147 – GET service with partial invocations, GBT and streaming, recovery of last block	362
Figure 148 – SET service with GBT, with server not supporting streaming, recovery of 3rd block.....	363
Figure 149 – ACTION-WITH-LIST service with bi-directional GBT and block recovery.....	364
Figure 150 – <b>Unconfirmed DataNotification service with GBT with partial invocation .....</b>	366
<b>Figure 151 – Confirmed DataNotification service with GBT .....</b>	367
<b>Figure 152 – DataNotification_Confirmed with GBT recovery .....</b>	368
Figure 153 – Identification/addressing scheme in the 3-layer, CO, HDLC based communication profile.....	408
Figure 154 – Summary of data link layer services .....	409
Figure 155 – Example: EventNotification triggered by the client .....	412
Figure 156 – Multi-drop configuration and its model.....	413
Figure 157 – Master/ Slave operation on the multi-drop bus.....	413
Figure 158 – Communication architecture .....	415
Figure 159 – Examples for lower-layer protocols in the TCP-UDP/IP based profile(s).....	416
Figure 160 – Identification / addressing scheme in the TCP-UDP/IP based profile(s) .....	417
Figure 161 – Summary of TCP / UDP layer services .....	419
<b>Figure 162 – The DLMS/COSEM CoAP communication profile .....</b>	423
<b>Figure 163 – CoAP transport layer primitives.....</b>	425
<b>Figure 164 – Mapping the DLMS ACSE service primitives to the CoAP-DATA service primitives .....</b>	427
<b>Figure 165 – Mapping of the xDLMS ASE service primitives to the CoAP-DATA service primitives .....</b>	428
Figure 166 – Communication architecture .....	430
Figure 167 – The DLMS/COSEM S-FSK PLC communication profile .....	431
Figure 168 – Co-existence of the connectionless and the HDLC based LLC sublayers .....	433
Figure 169 – Intelligent Search Initiator process flow chart .....	441
Figure 170 – The Discovery and Registration process .....	444
Figure 171 – MSC for the discovery and registration process .....	449
Figure 172 – MSC for successful confirmed AA establishment .....	450
Figure 173 – MSC for releasing an Application Association .....	451

Figure 174 – MSC for an EventNotification service .....	452
Figure 175 – MSC for the Discovery and Registration process .....	453
Figure 176 – MSC for successful confirmed AA establishment and the GET service .....	454
Figure 177 – Entities and interfaces of a smart metering system using the terminology of IEC 62056-1-0.....	457
Figure 178 – The DLMS/COSEM wired and wireless M-Bus communication profiles .....	458
Figure 179 – Summary of DLMS/COSEM M-Bus-based TL services .....	460
Figure 180 – Identification and addressing scheme in the wired M-Bus profile .....	465
Figure 181 – Link Layer Address for wireless M-Bus.....	466
Figure 182 – M-Bus TPDU formats.....	467
Figure 183 – CI <sub>TL</sub> without M-Bus data header .....	467
Figure 184 – M-Bus communication paths direct or cascaded .....	472
Figure 185 – Wired M-Bus frame structure, none M-Bus data header .....	473
Figure 186 – Wired M-Bus frame structure with long M-Bus data header .....	473
Figure 187 – Wireless M-Bus frame structure with short ELL, no M-Bus data header .....	474
Figure 188 – Wireless M-Bus frame structure with long ELL, no M-Bus data header .....	475
Figure 189 – Wireless M-Bus frame structure with long ELL and long M-Bus data header .....	475
Figure 190 – Daily billing data without / with DLMS/COSEM security applied .....	477
Figure 191 – MSC for the COSEM-OPEN service for wired M-Bus, none M-Bus header.....	478
Figure 192 – MSC the GET service for wired M-Bus, none M-Bus header.....	479
Figure 193 – Short wrapper .....	480
Figure 194 – LPWAN (SCHC) architecture outline .....	481
Figure 195 – The DLMS/COSEM LPWAN communication profile .....	482
Figure 196 – Wi-SUN Architecture (Layer 3 routing) .....	485
Figure 197 – Wi-SUN communication profile diagram .....	486
Figure 199 – General architecture with gateway .....	493
Figure 200 – The fields used for pre-fixing the COSEM APDUs .....	494
Figure 201 – Pull message sequence chart .....	495
Figure 202 – Push message sequence chart .....	496
Figure 203 – The DLMS/COSEM GET service on LPWAN.....	571
Figure C. 1 – MSC for key agreement using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme.....	586
Figure C. 2 – Ciphered xDLMS APDU protected by an ephemeral key established using the One-pass Diffie-Hellman (1e, 1s, ECC CDH) scheme .....	589
Figure C. 3 – Ciphered xDLMS APDU protected by an ephemeral key established using the Static Unified Model C(0e, 2s, ECC CDH) scheme.....	593
Figure D. 1 – Exchanging protected xDLMS APDUs between TP and server: example 1 .....	597
Figure D. 2 – Exchanging protected xDLMS APDUs between TP and server: example 2 .....	599
Table 1 – Client and server SAPs .....	48
Table 2 – Reserved wrapper port numbers in the UDP-based DLMS/COSEM TL .....	85
Table 3 – Reserved SAP numbers in the DLMS/COSEM CoAP communication profile .....	108
Table 4 – CoAP Request method codes .....	118

8/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
-------	------------	-----------------------	-----------------------

Table 5 – CoAP Success Response codes .....	118
Table 6 – CoAP Options used by the DLMS/COSEM CoAP TL .....	119
Table 7 – CoAP retransmission parameters .....	121
Table 8 – CoAP congestion control parameters .....	121
Table 9 – CoAP wrapper error response return. [Informative] .....	123
Table 10 – CoAP wrapper request/response context parameters .....	125
Table 11 – State transition table of the client side LLC sublayer .....	166
Table 12 – State transition table of the server side LLC sublayer .....	166
Table 13 – Table of reserved client addresses .....	169
Table 14 – Table of reserved server addresses .....	169
Table 15 – Handling inopportune address lengths .....	171
Table 16 – Control field bit assignments of command and response frames .....	171
Table 17 – Example for parameter negotiation values with the SNRM/UA frames .....	177
Table 18 – Summary of MAC addresses for the example .....	182
Table 19 – Broadcast UI frame handling .....	182
Table 20 – Clarification of the meaning of PDU size for DLMS/COSEM .....	202
Table 21 – Elliptic curves in DLMS/COSEM security suites .....	218
Table 22 – Ephemeral Unified Model key agreement scheme summary .....	222
Table 23 – One-pass Diffie-Hellman key agreement scheme summary .....	223
Table 24 – Static Unified Model key agreement scheme summary .....	225
Table 25 – <i>OtherInfo</i> subfields and substrings .....	226
Table 26 – Cryptographic algorithm ID-s .....	226
Table 27 – DLMS/COSEM security suites .....	227
Table 28 – Symmetric keys types .....	229
Table 29 – Key information with general-ciphering APDU and data protection .....	230
Table 30 – Asymmetric keys types and their use .....	232
Table 31 – X.509 v3 Certificate structure .....	236
Table 32 – X.509 v3 tbsCertificate fields .....	236
Table 33 – Naming scheme for the Root-CA instance (informative) .....	237
Table 34 – Naming scheme for the Sub-CA instance (informative) .....	237
Table 35 – Naming scheme for the end entity instance .....	238
Table 36 – X.509 v3 Certificate extensions .....	239
Table 37 – Key Usage extensions .....	240
Table 38 – Subject Alternative Name values .....	241
Table 39 – Issuer Alternative Name values .....	241
Table 40 – Basic constraints extension values .....	241
Table 41 – Certificates handled by DLMS/COSEM end entities .....	242
Table 42 – Security policy values (“Security setup” version 1) .....	247
Table 43 – Access rights values (“Association LN” ver 3 “Association SN” ver 4) .....	247
Table 44 – Ciphered xDLMS APDUs .....	248
Table 45 – Security control byte .....	250
Table 46 – Plaintext and Additional Authenticated Data .....	250

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	9/614
-----------------------	------------	-----------------------	-------

Table 47 – Use of the fields of the ciphering xDLMS APDUs .....	253
Table 48 – Example: glo-get-request xDLMS APDU.....	254
Table 49 – ACCESS service with general-ciphering, One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) key agreement scheme .....	256
Table 50 – DLMS/COSEM HLS authentication mechanisms .....	259
Table 51 – HLS example using authentication-mechanism5 with GMAC .....	260
Table 52 – HLS example using authentication-mechanism 7 with ECDSA.....	261
Table 53 – Codes for AL service parameters .....	264
Table 54 – Service parameters of the COSEM-OPEN service primitives .....	265
Table 55 – Service parameters of the COSEM-RELEASE service primitives .....	269
Table 56 – Service parameters of the COSEM-ABORT service primitives.....	272
Table 57 – Additional service parameters.....	274
Table 58 – Security parameters .....	275
Table 59 – APDUs used with security protection types.....	276
Table 60 – Service parameters of the GET service .....	278
Table 61 – GET service request and response types .....	278
Table 62 – Service parameters of the SET service.....	280
Table 63 – SET service request and response types.....	281
Table 64 – Service parameters of the ACTION service .....	283
Table 65 – ACTION service request and response types.....	284
Table 66 – Service parameters of the ACCESS service .....	289
Table 67 – Service parameters of the DataNotification service primitives.....	292
Table 68 – Service parameters of the EventNotification service primitives .....	293
Table 69 – Service parameters of the TriggerEventNotificationSending.request service primitive .....	294
Table 70 – Variable Access Specification .....	295
Table 71 – Service parameters of the Read service .....	295
Table 72 – Use of the Variable_Access_Specification variants and the Read.response choices	296
Table 73 – Service parameters of the Write service .....	299
Table 74 – Use of the Variable_Access_Specification variants and the Write.response choices	299
Table 75 – Service parameters of the UnconfirmedWrite service .....	301
Table 76 – Use of the Variable_Access_Specification variants .....	301
Table 77 – Service parameters of the InformationReport service .....	302
Table 78 – Service parameters of the SetMapperTable.request service primitives .....	303
Table 79 – Summary of ACSE services .....	303
Table 80 – Summary of xDLMS services .....	303
Table 81 – Functional Unit APDUs and their fields.....	308
Table 82 – COSEM application context names .....	311
Table 83 – COSEM authentication mechanism names .....	312
Table 84 – Cryptographic algorithm ID-s .....	312
Table 85 – xDLMS Conformance block.....	322
Table 86 – GET service types and APDUs.....	324
Table 87 – SET service types and APDUs.....	327

Table 88 – ACTION service types and APDUs .....	330
Table 89 – Mapping between the GET and the Read service.....	337
Table 90 – Mapping between the ACTION and the Read service .....	337
Table 91 – Mapping between the SET and the Write service .....	341
Table 92 – Mapping between the ACTION and the Write service .....	342
Table 93 – Mapping between the SET and the UnconfirmedWrite service .....	344
Table 94 – Mapping between the ACTION and the UnconfirmedWrite service .....	345
Table 95 – Mapping between the EventNotification and InformationReport services.....	346
Table 96 – GBT procedure state variables.....	352
Table 97 – xDLMS exception mechanism .....	369
Table 98 – Application associations and data exchange in the 3-layer, CO, HDLC based profile .....	410
Table 99 – Application associations and data exchange in the TCP-UDP/IP based profile ...	420
<b>Table 100 – Application associations and data exchange in the CoAP based communication profile .....</b>	<b>426</b>
Table 101 – Service parameters of the Discover service primitives .....	434
Table 102 – Service parameters of the Register service primitives .....	435
Table 103 – Service parameters of the PING service primitives.....	435
Table 104 – Service parameters of the RepeaterCall service primitives .....	437
Table 105 – Service parameters of the ClearAlarm service primitives .....	439
Table 106 – MAC addresses .....	447
Table 107 – Reserved IEC 61334-4-32 LLC addresses on the client side .....	447
Table 108 – Reserved IEC 61334-4-32 LLC addresses on the server side .....	447
Table 109 – Reserved HDLC based LLC addresses on the client side .....	448
Table 110 – Reserved HDLC based LLC addresses on the server side .....	448
Table 111 – Source and Destination APs and addresses of CI-PDUs .....	448
Table 112 – Application associations and data exchange in the S-FSK PLC profile using the connectionless LLC sublayer .....	450
Table 113 – Wired M-Bus Link Layer Addresses .....	465
Table 114 – DLMS/COSEM M-Bus-based TL $CI_{TL}$ values .....	466
Table 115 – CI fields used for link management purposes.....	468
Table 116 – Client and server SAPs.....	468
Table 117 – Application associations and data exchange in the M-Bus-based profiles .....	469
Table 118 – Example: Daily billing data.....	476
Table 119 – Reserved Application Process SAPs .....	480
Table 120 – Client and server SAPs.....	483
Table 121 – FANSPEC to Wi-SUN setup IC attribute mapping .....	489
Table 122 – Join states .....	489
Table 123 – UDP port numbering .....	490
Table 124 – Conformance block.....	498
Table 125 – A-XDR encoding the xDLMS InitiateRequest APDU .....	499
Table 126 – A-XDR encoding the xDLMS InitiateResponse APDU .....	500
Table 127 – BER encoding the AARQ APDU .....	503

Table 128 – The complete AARQ APDU .....	505
Table 129 – BER encoding the AARE APDU .....	506
Table 130 – The complete AARE APDU .....	510
Table 131 – A-XDR encoding of the xDLMS InitiateRequest APDU .....	511
Table 132 – Authenticated encryption of the xDLMS InitiateRequest APDU using service-specific global ciphering .....	512
Table 133 – BER encoding of the AARQ APDU .....	513
Table 134 – A-XDR encoding of the xDLMS InitiateResponse APDU using service-specific global ciphering.....	514
Table 135 – Authenticated encryption of the xDLMS InitiateResponse APDU .....	515
Table 136 – BER encoding of the AARE APDU.....	515
Table 137 – BER encoding of the RLRQ APDU .....	517
Table 138 – BER encoding of the RLRE APDU.....	517
Table 139 – The objects used in the examples .....	531
Table 140 – Example: Reading the value of a single attribute without block transfer .....	532
Table 141 – Example: Reading the value of a list of attributes without block transfer .....	533
Table 142 – Example: Reading the value of a single attribute with block transfer .....	534
Table 143 – Example: Reading the value of a list of attributes with block transfer .....	537
Table 144 – Example: Writing the value of a single attribute without block transfer .....	540
Table 145 – Example: Writing the value of a list of attributes without block transfer .....	541
Table 146 – Example: Writing the value of a single attribute with block transfer .....	542
Table 147 – Example: Writing the value of a list of attributes with block transfer .....	545
Table 148 – Example: ACCESS service without block transfer .....	548
Table 149 – Profile generic buffer – get-response with normal encoding.....	555
Table 150 – Profile generic buffer – get-response with null-data compression .....	558
Table 151 – Profile generic buffer – get-response with compact-array encoding .....	561
Table 152 – Profile generic buffer – Get-response with null-data and delta-value encoding ..	564
Table 153 – Comparison of various encoding methods for get-response APDU .....	568
Table 154 – Combination of the various encoding methods and V.44 compression for get-response APDU.....	568
Table 155 – Get service example.....	569
Table 156 – Data-Notification service with Profile generic.....	572
Table 157 – ACCESS service .....	576
Table A. 1 – ECC_P256_Domain_Parameters .....	581
Table A. 2 – ECC_P384_Domain_Parameters .....	581
Table B. 1 – Fields of public key Certificates using P-256 signed with P-256 .....	583
Table C. 1 – Test vector for key agreement using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme.....	587
Table C. 2 – Test vector for key agreement using the One-pass Diffie-Hellman (1e, 1s, ECC CDH) scheme.....	590
Table C. 3 – Test vector for key agreement using the Static-Unified Model (0e, 2s, ECC CDH) scheme .....	594

## Foreword

### Copyright

© Copyright 1997-2021 DLMS User Association.

This Edition 11 of the Green Book specifies important new elements:

- CoAP based transport layer protocol specified in 7.3 and CoAP based communication profile specified in 10.4;
- GBT as specified in 9.4.6.13 is extended to specify its use for transporting confirmed DataNotification services.

In addition, some editorial changes have been made. See the List of main changes.

This Technical Report is confidential. It may not be copied, nor handed over to persons outside the standardisation environment.

The copyright is enforced by national and international law. The "Berne Convention for the Protection of Literary and Artistic Works" which is signed by 173 countries worldwide and other treaties apply.

### Liability

DLMS User Association Publications have the form of recommendations for international use. While all reasonable efforts are made to ensure that the technical content of DLMS User Association Publications is accurate, the DLMS User Association cannot be held responsible for the way in which they are used or for any misinterpretation by any end user.

No liability shall attach to DLMS User Association or its directors, employees, servants or agents including individual experts and members of its technical 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 DLMS User Association Publication or any other DLMS User Association Publications.

### Acknowledgement

This document has been established by the WG Maintenance of the DLMS UA.

Clause 9.2, Information security in DLMS/COSEM is based on parts of NIST documents. Reprinted courtesy of the National Institute of Standards and Technology, Technology Administration, U.S. Department of Commerce. Not copyrightable in the United States.

### Status of standardisation

The content of Green Book Edition 10 is in line with IEC 62056-5-3:2021 Ed. 4.0, Electricity metering data exchange – The DLMS/COSEM suite – Part 5-3: DLMS/COSEM application layer.

To bring the changes in Green Book Edition 11 to international standardisation, updates to IEC 62056-5-3:2021 will be initiated by IEC TC13 WG14.

IEC TC13 WG14 will standardise the profile for CoAP in the IEC 62056 series.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	13/614
-----------------------	------------	-----------------------	--------

## Revision history

1. Version	Date	Author	Comment
Release 1	1 <sup>st</sup> April 1998	DLMS UA	Initial version
First Edition	1 <sup>st</sup> May 2000	DLMS UA	Major rework, adapted to CDVs of IEC TC13
Second Edition	15 <sup>th</sup> May 2001	DLMS UA	Considering comments to CDVs by IEC National Committees
Third Edition	30 <sup>th</sup> March 2002	DLMS UA	Content adapted to IEC International Standards
Fourth Edition	15 <sup>th</sup> April 2004	DLMS UA	Major rework, adapted to EN and to CDs and NP of IEC TC13 (chapters 4 and 7 are new, change marks added to others)
Fifth Edition	26 <sup>th</sup> August 2005	DLMS-UA	Content aligned with IEC TC 13 CDV-s and comments received.
Sixth Edition	27 <sup>th</sup> August 2007	DLMS UA	Technical content aligned with IEC TC 13 standards published in 2006 / 2007 Document restructured. See list of changes. Sent to WG for approval.
Seventh Edition	22 <sup>nd</sup> December 2009	DLMS UA	Includes: <ul style="list-style-type: none"><li>- SN block transfer</li><li>- Data security</li><li>- S-FSK PLC profile</li></ul>
Eighth Edition	4 <sup>th</sup> July 2014	DLMS UA	Includes: <ul style="list-style-type: none"><li>- Security extensions;</li><li>- Compression;</li><li>- ACCESS service;</li><li>- DataNotification service;</li><li>- General block transfer mechanism;</li><li>- XML schema;</li><li>- Wired and wireless M-Bus profile;</li><li>- SMS profile;</li><li>- Gateway protocol;</li><li>- Compact array encoding example.</li></ul>
Eighth Edition corrected	7 <sup>th</sup> July 2014	DLMS UA	Wrong Figure 139 replaced with the correct one. Missing CIASE APDU module added as 10.5.9.
Edition 8.0 Corrigendum 1	14 <sup>th</sup> December	DLMS UA	Technical and editorial Corrigendum 1 to Edition 8.0.
Edition 8.1	14th <sup>th</sup> December 2015	DLMS UA	Consolidated edition integrating Corrigendum 1.
Edition 8.2	19 <sup>th</sup> January 2017	DLMS UA	Editorial corrections. Test vector corrections. In line with IEC 62056-5-3 Ed.3.0:2017
Edition 8.3	30 <sup>th</sup> June 2017	DLMS UA	Editorial corrections. Clause added describing the use of the ConfirmedServiceError and ExceptionResponse APDUs. Extended specification of the ExceptionResponse APDU included in Abstract Syntax & XML Schema
Edition 9	8 <sup>th</sup> May 2019	DLMS UA	See below.
Edition 10	31 <sup>st</sup> August 2020	DLMS UA	See below.

## List of main technical changes in Edition 9

<b>Item</b>	<b>Clause</b>	<b>Change</b>
1.	9.1.4.4.5	Substantively replaced for clarity. Description of Block Transfer mechanisms
2.	9.1.4.4.9	Substantively replaced for clarity. Description of General Block Transfer.
3.	9.3.2	Inclusion of system title for HLS mechanisms as needed in Calling_AP_Title
4.	9.3.7	Clarification regarding response to SET.request with Attr.0
5.	9.4.6.4	Addition/Correction of response APDU in table 79
6.	9.4.6.13.2	New content clarifying the procedure of operation of the General Block Transfer (GBT)
7.	9.4.6.13.3	New content clarifying the state variables of the General Block Transfer (GBT)
8.	9.4.6.13.4	New content clarifying the stream sub-procedure of the General Block Transfer (GBT)
9.	9.4.6.13.5	New content clarifying the APDU processing sub-procedure of the General Block Transfer (GBT)
10.	9.4.6.13.6	New content clarifying the retry sub-procedure of the General Block Transfer (GBT)
11.	9.4.6.14	Paragraph moved for clarity
12.	12.2	Clarification of the example in Table 132
13.	C.1	Additions of brackets to key agreement data for clarity to Figure C.1
14.	C.2	Addition of bracket to key-info data for clarity to Figure C.2

## List of main technical changes in Edition 10

<b>Item</b>	<b>Clause</b>	<b>Change</b>
1.	9.1.4.3.1	Push added
2.	9.3.10	Text changed to add reliable push
3.	9.4.6.2.2	New clause for push services
4.	9.4.6.7	New description and figures for the reliable push
5.	9.5	Delta types added, syntax added for confirm push
6.	9.6	Delta types added, XML Schema added for confirm push
7.	10.8	New profile for LPWAN
8.	10.9	New profile added for Wi-SUN
9.	14.4	Profile generic IC buffer attribute encoding examples using various encoding techniques added
10.	Figure 5	Confirmed DataNotification service added to Figure
11.	Figure 88	Push added
12.	Figure 90	Push added
13.	Figure 91	Push added
14.	Figure 114	Notification services added for reliable push
15.	Figure 115	Notification services added for reliable push
16.	Table 67	.response and .confirm parameters added
17.	Table 80	Add services for reliable push

### List of main technical changes in Edition 11

Item	Clause	Change
1.	9.3.10	Data notification service modifies to support CoAP reliable and unreliable push
2.	10.4	New clause covering the CoAP profile. Further modifications in clauses 4.8, and 7 to support CoAP
3.	7.3.6	New clause to support CoAP.
4.		
5.		
6.		
7.		
8.		
9.		
10.		
11.		
12.		
13.		
14.		
15.		
16.		
17.		

## 1 Scope

The DLMS/COSEM specification specifies an interface model and communication protocols for data exchange with **connected devices**.

The interface model provides a view of the functionality of the **device** as it is available at its interface(s). It uses generic building blocks to model this functionality. The model does not cover internal, implementation-specific issues.

Communication protocols define how the data can be accessed and transported.

The DLMS/COSEM specification follows a three-step approach as illustrated in

Figure 1:

- Step 1, Modelling: This covers the interface model of a **device** and rules for data identification;
- Step 2, Messaging: This covers the services for mapping the interface model to protocol data units (APDU) and the encoding of this APDUs.
- Step 3, Transporting: This covers the transportation of the messages through the communication channel.

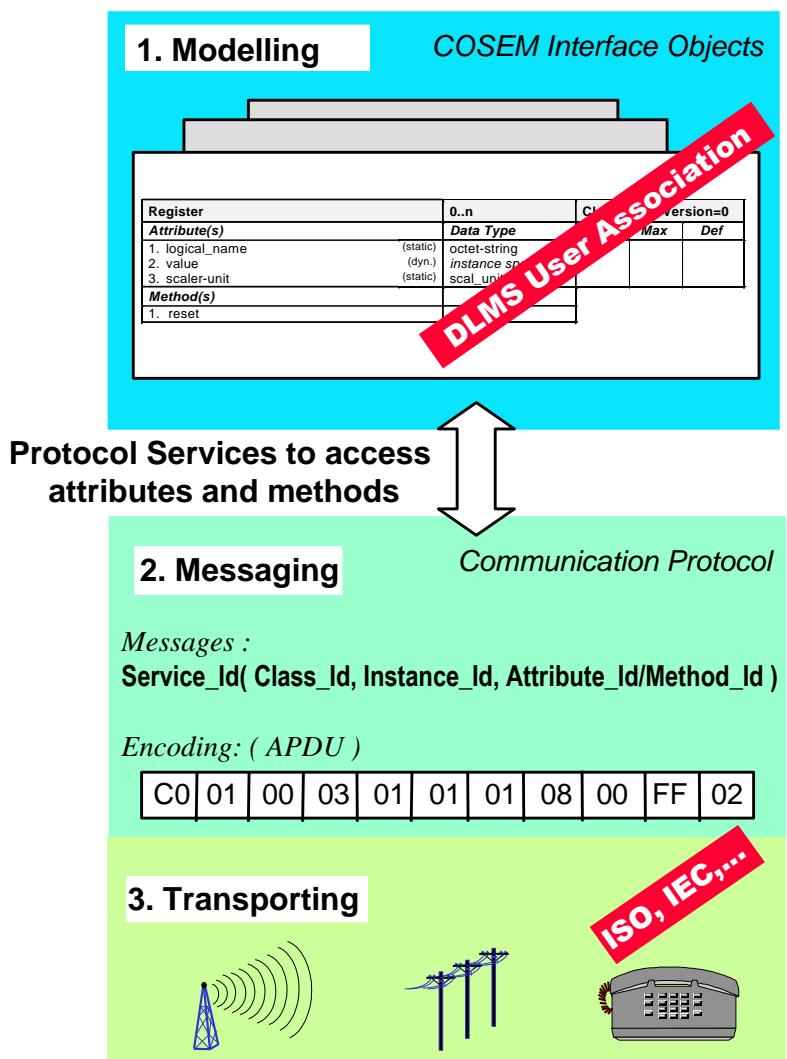


Figure 1 – The three steps approach of COSEM: Modelling – Messaging – Transporting

Step 1 is specified in the document "COSEM interface classes and the OBIS identification system" DLMS UA 1000-1. It specifies the COSEM interface classes, the OBIS identification system used to identify instances of these classes, called interface objects, and the use of interface objects for modelling the various functions of the **device**.

Step 2 and 3 are specified in this Technical Report.

The DLMS/COSEM application layer (AL) specifies the services to establish logical connections between a client and (a) server(s) and the services to access attributes and methods of the COSEM objects. The DLMS/COSEM AL is specified in Clause 9.

DLMS/COSEM communication media specific profiles specify how application layer messages can be transported over various communication media. Each communication profile specifies the set of the protocol layers required to support the DLMS/COSEM AL on top. See also 4.8.

Large scale deployment of smart connected systems requires strong information security mechanisms to protect the privacy of consumers, the business interests of the service providers and the security of the infrastructure.

DLMS/COSEM provides built-in security mechanisms from the outset. Initially, it provided mechanisms for the identification and authentication of clients and servers, as well as specific access rights to COSEM object attributes and methods within application associations (AAs) established between a client and a server. Ciphered APDUs were also available to allow protecting the messages exchanged between clients and servers.

In the next step, the details of ciphering using symmetric key algorithms, providing authentication and encryption as well as key transport mechanisms have been specified.

Rules for conformance testing are specified in the document DLMS UA 1001-1 "DLMS/COSEM Conformance Test Process".

Terms are explained in Clause 3 and in DLMS UA 1002 "COSEM Glossary of Terms".

18/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
--------	------------	-----------------------	-----------------------

## 2 Normative references

Ref.	Title
DLMS UA 1000-1 Part 2 Ed.15:2021	COSEM Interface Classes and OBIS Identification System, the "Blue Book"
DLMS UA 1000-1 Part 1	COSEM Interface Classes and OBIS Identification System, the "Blue Book" NOTE This undated reference is used unless a specific clause needs to be referenced.
DLMS UA 1001-1	DLMS/COSEM Conformance test and certification process, the "Yellow Book"
DLMS UA 1002 Ed. 1.0:2003	COSEM Glossary of Terms, "White Book"
IEC 61334-4-1:1996	Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 1: Reference model of the communication system
IEC 61334-4-32:1996	Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 32: Data link layer – Logical link control (LLC)
IEC 61334-4-41:1996	Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 41: Application protocol – Distribution line message specification
IEC 61334-4-511:2000	Distribution automation using distribution line carrier systems – Part 4-511: Data communication protocols – Systems management – CIASE protocol
IEC 61334-5-1:2001	Distribution automation using distribution line carrier systems – Part 5-1: Lower layer profiles – The spread frequency shift keying (S-FSK) profile
IEC 61334-6:2000	Distribution automation using distribution line carrier systems – Part 6: A-XDR encoding rule
IEC 62056-1-0	Electricity metering data exchange – The DLMS/COSEM suite – Part 1 0: Smart metering standardisation framework
IEC 62056-21:2002	Electricity metering – Data exchange for meter reading, tariff and load control – Part 21: Direct local data exchange
IEC 62056-8-20:2016	Electricity metering data exchange - The DLMS/COSEM suite - Part 8-20: Mesh communication profile for neighbourhood networks
ISO/IEC 7498-1:1994	Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model
ISO/IEC 8649 Ed. 2.0:1996	Information technology – Open Systems Interconnection – Service definition for the Association Control Service Element NOTE This standard has been replaced by ISO/IEC 15953:1999
ISO/IEC 8650-1 Ed 2.0:1996	Information technology – Open systems interconnection – Connection-oriented protocol for the association control service element: Protocol specification NOTE This standard has been replaced by ISO/IEC 15954:1999
ISO/IEC 8802-2 Ed. 3.0:1998	Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 2: Logical link control
ISO/IEC 8824:2008	Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation
ISO/IEC 8825-1:2015	Information technology - ASN.1 encoding rules: Specification of : Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).
ISO/IEC 13239:2002	Information Technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures
ISO/IEC 15953:1999	Information technology — Open Systems Interconnection — Service definition for the Application Service Object Association Control Service Element NOTE This standard cancels and replaces ISO/IEC 8649:1996 and its Amd. 1:1997 and Amd. 2:1998, of which it constitutes a technical revision.
ISO/IEC 15954:1999	Information technology — Open Systems Interconnection — Connection-mode protocol for the Application Service Object Association Control Service Element

## DLMS/COSEM Architecture and Protocols

	NOTE This standard cancels and replaces ISO/IEC 8650-1:1999 and its Amd. 1:1997 and Amd. 2:1998, of which it constitutes a technical revision.
EN13757-1:2014	Communication system for and remote reading of meters – Part 1: Data exchange
EN 13757-2:2004	Communication system for and remote reading of meters – Part 2 : Physical and Link Layer, Twisted Pair Baseband (M-Bus)
EN 13757-3:2018	Communication systems for meters - Part 3: Application protocols
EN 13757-4:2013	Communication system for and remote reading of meters – Part 4: Wireless meter (Radio meter reading for operation in SRD bands)
EN 13757-5:2015	Communication system for and remote reading of meters – Part 5: Wireless relaying
EN 13757-6:2015	Communication system for meters – Part 6: Local Bus
EN 13757-7:2018	Communication systems for meters - Part 7: Transport and security services
ETSI-TS-102-887-2	Electromagnetic compatibilityand Radio spectrum Matters (ERM); Short Range Devices; Smart Metering Wireless Access Protocol; Part 2: Data Link Layer (MAC Sub-layer)
IEEE 802.1ar	IEEE Standard for Local and Metropolitan Area Networks – Secure Device Identity, IEEE Std 802.1AR-2009
IEEE 802.1X	IEEE Standard for Local and Metropolitan Area Networks – Port Based Network Access Control”, IEEE Std 802.1X-2010
IEEE 802.11i	IEEE Standard for Information Technology— Telecommunications and information exchange between systems Local and metropolitan area networks—Specific requirements Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications IEEE Std 802.11-2012  NOTE the 802.11i amendment was specifically for the inclusion of Wi-Fi Protected Access (WPA 2) security which is the part that is relevant to this standard.
IEEE 802.15.4	IEEE Standard for Low-Rate Wireless Networks
ITU-T V.44: 2000	SERIES V: DATA COMMUNICATION OVER THE TELEPHONE NETWORK – Error control – V.44:2000, Data compression procedures
ITU-T X.211	SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY – Information technology – Open systems interconnection – Physical Service Definition
ITU-T X.509:2008	SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY – Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks
ITU-T X.693 (11/2008)	Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)
ITU-T X.693 Corrigendum 1(10/2011)	Information technology – ASN.1 encoding rules: XML Encoding Rules (XER) Technical Corrigendum 1
ITU-T X.694 (11/2008)	Information technology – ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1
ITU-T X.694 Corrigendum 1 (10/2011)	Information technology – ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1Technical Corrigendum 1
ANSI C12.21:1999	Protocol Specification for Telephone Modem Communication
FIPS PUB 180-4:2012	Secure Hash Standard (SHS)
FIPS PUB 186-4:2013	Digital Signature Standard (DSS)
FIPS PUB 197:2001	Advanced Encryption Standard (AES)
LoRaWAN Spec 1.0.3	<a href="https://lora-alliance.org/resource-hub/lorawanr-specification-v103">https://lora-alliance.org/resource-hub/lorawanr-specification-v103</a> .
NIST SP 800-21:2005	Guideline for Implementing Cryptography in the Federal Government
NIST SP 800-32:2001	Introduction to Public Key Technology and the Federal PKI Infrastructure
NIST SP 800-38D:2007	Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC
NIST SP 800-56A Rev. 2: 2013	Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography
NIST SP 800-57:2012	Recommendation for Key Management – Part 1: General (Revision 3)
NSA1	Suite B Implementer's Guide to FIPS 186-3 (ECDSA), Feb 3rd 2010

## DLMS/COSEM Architecture and Protocols

NSA2	Suite B Implementer's Guide to NIST SP800-56A, 28th July 2009
NSA3	NSA Suite B Base Certificate and CRL Profile, 27th May 2008
[FANSPEC]	Wi-SUN Alliance: Field Area Network Working Group (FANWG):Technical Profile Specification:Field Area Network:Version 1v26.
[PHYSPEC]	Wi-SUN Alliance: PHY Working Group (PHYWG) Wi-SUN PHY Specification Revision 1V02
ANSI/TIA-4957.200	Layer 2 Standard Specification for the Smart Utility Network
ANSI/TIA 4957.210	Multi-Hop Sublayer Specification-Extension on Field Area Networks
The following RFCs are available on line from the Internet Engineering Task Force (IETF): <a href="https://www.ietf.org/rfc/std-index.txt">https://www.ietf.org/rfc/std-index.txt</a> , <a href="https://www.ietf.org/rfc/">https://www.ietf.org/rfc/</a>	
RFC 768	User Datagram Protocol
RFC 793	Transmission Control Protocol
RFC 1213	Management Information Base for Network Management of TCP/IP-based internets: MIB-II
RFC 2460	Internet Protocol, Version 6
RFC 3315	Dynamic Host Configuration Protocol for IPv6 (DHCPv6)
RFC 3394	Advanced Encryption Standard (AES) Key Wrap Algorithm, 2002
RFC 4108	Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages, 2005
RFC 4291	IP Version 6 Addressing Architecture
RFC 4443	Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification
RFC 4944	Transmission of IPv6 Packets over IEEE 802.15.4 Networks
RFC 5280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008
RFC 5216	The EAP-TLS Authentication Protocol
RFC 5280	Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile
RFC 6206	The Trickle Algorithm
RFC 6282	Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks
RFC 6550	RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks
RFC 6775	Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks 409 (6LoWPANs)
RFC 7217	A Method for Generating Semantically Opaque Interface Identifiers with IPv6 Stateless Address Autoconfiguration (SLAAC). Edited by F. Gont.
RFC 7731	Multicast Protocol for Low-Power and Lossy Networks (MPL)
<b>RFC 7252</b>	<b>The Constrained Application Protocol.</b>
RFC 7774	Multicast Protocol for Low-Power and Lossy Networks (MPL) Parameter Configuration Option for DHCPv6
<b>RFC 7959</b>	<b>Block-Wise Transfers in the Constrained Application Protocol (CoAP)</b>
RFC 8376	Low-Power Wide Area Network (LPWAN) Overview. Edited by S. Farrell, May 2018.
RFC 8724	SCHC – Generic Framework for Static Context Header Compression and Fragmentation. April 2020.
IETF Internet Draft	Static Context Header Compression (SCHC) over LoRaWAN.
STD0005 (1981)	Internet Protocol. Also: RFC0791, RFC0792, RFC0919, RFC0922, RFC0950, RFC1112
STD0006 (1980)	User Datagram Protocol. Also: RFC0768
STD0007 (1981)	Transmission Control Protocol. Also: RFC0793

### 3 Terms, definitions and abbreviations and symbols

#### 3.1 General DLMS/COSEM definitions

##### 3.1.1

**ACSE APDU**

APDU used by the Association Control Service Element (ACSE)

##### 3.1.2

**application association**

cooperative relationship between two application entities, formed by their exchange of application protocol control information through their use of presentation services

##### 3.1.3

**application context**

set of application service elements, related options and any other information necessary for the interworking of application entities in an application association

##### 3.1.4

**application entity**

the system-independent application activities that are made available as application services to the application agent, e.g., a set of application service elements that together perform all or part of the communication aspects of an application process

##### 3.1.5

**application process**

an element within a real open system which performs the information processing for a particular application

[SOURCE: ISO/IEC 7498-1:1994, 4.1.4]

##### 3.1.6

**authentication mechanism**

the specification of a specific set of authentication-function rules for defining, processing, and transferring authentication-values

[SOURCE: ISO/IEC 15953:1999, 3.5.11]

##### 3.1.7

**block**

one portion of an xDLMS APDU; the payload of a GBT APDU

##### 3.1.8

**client**

application process running in the data collection system

##### 3.1.9

**client/server**

relationship between two computer programs in which one program, the client, makes a service request from another program, the server, which fulfils the request

##### 3.1.10

**confirmed GBT procedure**

procedure in which the sender sends streams of GBT APDUs and at the end of each stream the recipient acknowledges the blocks received and attempts recovering any missing blocks

Note 1 to entry: A GBT stream consists of one or more GBT APDUs.

22/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
--------	------------	-----------------------	-----------------------

Note 2 to entry: In the case of a confirmed GBT stream the end of the stream is indicated by the streaming bit to set to FALSE (0). In the case of an unconfirmed GBT stream the end of the stream is indicated by the Final bit set to TRUE (1).

### 3.1.11

#### **COSEM**

Comprehensive Semantic Model for Energy Management; refers to the COSEM object model

### 3.1.12

#### **COSEM APDU**

comprises ACSE APDUs and xDLMS APDUs

### 3.1.13

#### **COSEM data**

COSEM object attribute values, method invocation and return parameters

### 3.1.14

#### **COSEM interface class**

entity with specific set of attributes and methods modelling a certain function on its own or in relation with other COSEM interface classes

### 3.1.15

#### **COSEM object**

instance of a COSEM interface class

### 3.1.16

#### **DLMS/COSEM**

refers to the application layer providing xDLMS services to access COSEM interface object attributes. Also refers to the DLMS/COSEM Application layer and the COSEM data model together.

### 3.1.17

#### **DLMS context**

a specification of the service elements of DLMS and semantics of communication to be used during the lifetime of an application association

[SOURCE: IEC 61334-4-41:1996, 3.3.5]

### 3.1.18

#### **entity authentication**

corroboration that an entity is the one claimed

[SOURCE: ISO/IEC 9798-1:2010, 3.14]

### 3.1.19

#### **gap**

empty space i.e. missing blocks in the receive queue RQ

Note to entry: A receive queue RQ may have one or more gaps. In each gap, one or more blocks may be missing.

### 3.1.20

#### **GBT APDU**

xDLMS APDU with control information that carries a block of another xDLMS APDU or an empty block

### 3.1.21

#### **GBT exchange**

exchanging GBT APDUs that carry the service primitives of the same service

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	23/614
-----------------------	------------	-----------------------	--------

**3.1.22****GBT stream**

a sequence of GBT APDUs

**3.1.23****general block transfer****GBT**

DLMS/COSEM application layer mechanism that can transfer any other xDLMS APDU that would be otherwise too long to fit into the maximum APDU size negotiated, in several blocks.

Note to entry: GBT can be forced by including GBT parameters in the .request service primitive.

**3.1.24****logical device**

abstract entity within a physical device, representing a subset of the functionality modelled with COSEM objects

**3.1.25****master**

central station – station which takes the initiative and controls the data flow

**3.1.26****message**

xDLMS APDU carrying a service primitive in an encoded form, which may also be cryptographically protected

**3.1.27****mutual authentication**

entity authentication which provides both entities with assurance of each other's identity

Note to entry: The DLMS/COSEM HLS authentication mechanism provides mutual authentication.

[SOURCE: ISO/IEC 9798-1:2010, 3.18 modified by adding Note 1]

**3.1.28****overflow**

more GBT APDUs received in one stream than the size of the GBT window

**3.1.29****physical device**

the highest level element used in the COSEM interface model of **devices**

**3.1.30****pull operation**

style of communication where the request for a given transaction is initiated by the client

**3.1.31****push operation**

style of communication where the request for a given transaction is initiated by the server

**3.1.32****receive queue****RQ**

placeholder for the blocks of the APDU received in a GBT stream

**3.1.33****server**

an application process running in a **device**

24/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
--------	------------	-----------------------	-----------------------

**3.1.34****send queue****SQ**

placeholder for the blocks of the APDU to be sent

**3.1.35****service-specific block transfer**

DLMS/COSEM application layer mechanism that can transfer an xDLMS APDU corresponding to a specific service primitive, that would be otherwise too long to fit into the maximum APDU size negotiated, in several blocks

**3.1.36****streaming window**

number of GBT APDUs that can be received in a stream

**3.1.37****slave**

station responding to requests of a master station.

Note to entry: A **device** is normally a slave station.**3.1.38****system title**

unique identifier of the system

**3.1.39****unconfirmed GBT procedure**

procedure in which the sender sends and the recipient receives a single stream of GBT APDUs, the recipient does not acknowledge the blocks received and does not attempt to recover any blocks lost

Note to entry: This is used to carry unconfirmed service requests from the client to the server or unsolicited service requests from the server to the client.

**3.1.40****unilateral authentication**

entity authentication which provides one entity with assurance of the other's identity but not vice versa

Note to entry: The DLMS/COSEM LLS authentication mechanism provides unilateral authentication.

[SOURCE: ISO/IEC 9798-1:2010, 3.39]

**3.1.41****xDLMS**

extended DLMS; refers to the DLMS protocol with the extensions specified in this Technical Report

**3.1.42****xDLMS APDU**

APDU used by the xDLMS Application Service Element (xDLMS ASE)

**3.1.43****xDLMS message**

xDLMS APDU exchanged between a client and a server or between a third party and a server

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	25/614
-----------------------	------------	-----------------------	--------

### 3.2 Definitions related to cryptographic security

#### 3.2.1

**access control**

restricts access to resources to only privileged entities.

[SOURCE: NIST SP 800-57:2012, Part 1]

#### 3.2.2

**asymmetric key algorithm**

see Public key cryptographic algorithm

#### 3.2.3

**authentication**

a process that establishes the source of information, provides assurance of an entity's identity or provides assurance of the integrity of communications sessions, messages, documents or stored data.

[SOURCE: NIST SP 800-57:2012, Part 1]

#### 3.2.4

**authentication code**

a cryptographic checksum based on an approved security function (also known as a Message Authentication Code)

[SOURCE: NIST SP 800-57:2012, Part 1]

#### 3.2.5

**certificate**

see public key certificate

#### 3.2.6

**Certification Authority****CA**

the entity in a Public Key Infrastructure (PKI) that is responsible for issuing public key certificates and exacting compliance to a PKI policy

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

#### 3.2.7

**Certificate Policy****CP**

a specialized form of administrative policy tuned to electronic transactions performed during certificate management. A Certificate Policy addresses all aspects associated with the generation, production, distribution, accounting, compromise recovery, and administration of digital certificates. Indirectly, a certificate policy can also govern the transactions conducted using a communications system protected by a certificate-based security system. By controlling critical certificate extensions, such policies and associated enforcement technology can support provision of the security services required by particular applications.

[SOURCE: NIST SP 800-32:2001]

#### 3.2.8

**challenge**

a time variant parameter generated by a verifier

[SOURCE: ITU-T X.811:1995, 3.8]

**3.2.9****ciphering**

authentication and / or encryption using symmetric key algorithms

**3.2.10****ciphertext**

data in its encrypted form

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.11****cofactor**

the order of the elliptic curve group divided by the (prime) order of the generator point (i.e. the base point) specified in the domain parameters

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

**3.2.12****confidentiality**

the property that sensitive information is not disclosed to unauthorized entities

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.13****cryptographic algorithm**

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

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.14****cryptographic key****key**

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

Note to entry:

Examples include:

1. The transformation of plaintext data into ciphertext data,
2. The transformation of ciphertext data into plaintext data,
3. The computation of a digital signature from data,
4. The verification of a digital signature,
5. The computation of an authentication code from data,
6. The verification of an authentication code from data and a received authentication code,
7. The computation of a shared secret that is used to derive keying material.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.15****cryptoperiod**

the time span during which a specific key is authorized for use or in which the keys for a given system or application may remain in effect

[SOURCE: NIST SP 800-57:2012, Part 1]

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	27/614
-----------------------	------------	-----------------------	--------

**3.2.16****dedicated key**

in DLMS/COSEM, a symmetric key used within a single instance of an Application Association. See also session key

**3.2.17****deprecated**

not recommended for new implementations

**3.2.18****digital signature**

the result of a cryptographic transformation of data that, when properly implemented with supporting infrastructure and policy, provides the services of:

1. origin authentication
2. data integrity, and
3. signer non-repudiation

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.19****directly trusted CA**

a directly trusted CA is a CA whose public key has been obtained and is being stored by an end entity in a secure, trusted manner, and whose public key is accepted by that end entity in the context of one or more applications

[SOURCE: ISO/IEC 15945:2002, 3.4]

**3.2.20****directly trusted CA key**

a directly trusted CA key is a public key of a directly trusted CA. It has been obtained and is being stored by an end entity in a secure, trusted manner. It is used to verify certificates without being itself verified by means of a certificate created by another CA.

Note to entry: Directly trusted CAs and directly trusted CA keys may vary from entity to entity. An entity may regard several CAs as directly trusted CAs.

[SOURCE: ISO/IEC 15945:2002, 3.5]

**3.2.21****distribution**

see key distribution

**3.2.22****domain parameters**

the parameters used with a cryptographic algorithm that are common to a domain of users

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

**3.2.23****encryption**

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

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.24****ephemeral key**

a cryptographic key that is generated for each execution of a key establishment process and that meets other requirements of the key type (e.g., unique to each message or session). In some cases ephemeral keys are used more than once, within a single “session” (e.g., broadcast applications) where the sender generates only one ephemeral key pair per message and the private key is combined separately with each recipient’s public key.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.25****global key**

a key that is intended for use for a relatively long period of time and is typically intended for use in many instances of a DLMS/COSEM Application Association, see also static symmetric key

**3.2.26****hash function**

a function that maps a bit string of arbitrary length to a fixed-length bit string. Approved hash functions satisfy the following properties:

- 1) One-way: It is computationally infeasible to find any input that maps to any pre-specified output, and
- 2) Collision resistant: It is computationally infeasible to find any two distinct inputs that map to the same output.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.27****hash value**

the result of applying a hash function to information

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.28****initialization vector****IV**

a vector used in defining the starting point of a cryptographic process

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.29****identification**

the process of verifying the identity of a user, process, or device, usually as a prerequisite for granting access to resources in an IT system

[SOURCE: NIST SP 800-47:2002]

**3.2.30****key**

see cryptographic key

**3.2.31****key agreement**

a (pair-wise) key-establishment procedure in which the resultant secret keying material is a function of information contributed by both participants, so that neither party can predetermine the value of the secret keying material independently from the contributions of the other party. Contrast with key-transport.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	29/614
-----------------------	------------	-----------------------	--------

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

### 3.2.32

#### **key-confirmation**

a procedure to provide assurance to one party (the key-confirmation recipient) that another party (the key-confirmation provider) actually possesses the correct secret keying material and/or shared secret

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

### 3.2.33

#### **key-derivation function**

a function by which keying material is derived from a shared secret (or a key) and other information

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

### 3.2.34

#### **key distribution**

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

[SOURCE: NIST SP 800-57:2012, Part 1]

### 3.2.35

#### **key-encrypting key**

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

Note to entry: In DLMS/COSEM it is the master key.

[SOURCE: NIST SP 800-57:2012 Part 1, modified by adding the Note]

### 3.2.36

#### **key establishment**

the procedure that results in keying material that is shared among different parties

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

### 3.2.37

#### **key pair**

a public key and its corresponding private key; a key pair is used with a public key algorithm

[SOURCE: NIST SP 800-57:2012, Part 1]

### 3.2.38

#### **key revocation**

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

[SOURCE: NIST SP 800-57:2012, Part 1]

### 3.2.39

#### **key-transport**

a (pair-wise) key-establishment procedure whereby one party (the sender) selects a value for the secret keying material and then securely distributes that value to another party (the receiver). Contrast with key agreement.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

30/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
--------	------------	-----------------------	-----------------------

**3.2.40****key wrapping**

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

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.41****message authentication code****MAC**

a cryptographic checksum on data that uses a symmetric key to detect both accidental and intentional modifications of data

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.42****message digest**

the result of applying a hash function to a message. Also known as "hash value".

[SOURCE: FIPS PUB 186-4:2013]

**3.2.43****named curve**

a set of ECDH domain parameters is also known as a "curve". A curve is a "named curve" if the domain parameters are well known and defined and can be identified by an Object Identifier; otherwise, it is called a "custom curve".

[SOURCE: RFC 5349]

**3.2.44****nonce**

a time-varying value that has at most an acceptably small chance of repeating. For example, the nonce may be a random value that is generated anew for each use, a timestamp, a sequence number, or some combination of these.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

**3.2.45****non-repudiation**

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

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.46****password**

a string of characters (letters, numbers and other symbols) that are used to authenticate an identity or to verify access authorization or to derive cryptographic keys.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.47****plaintext**

intelligible data that has meaning and can be understood without the application of decryption

[SOURCE: NIST SP 800-57:2012, Part 1]

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	31/614
-----------------------	------------	-----------------------	--------

**3.2.48****private key**

a cryptographic key, used with a public key cryptographic algorithm, which is uniquely associated with an entity and is not made public. In an asymmetric (public) cryptosystem, the private key is associated with a public key. Depending on the algorithm, the private key may be used, for example, to:

- 1) Compute the corresponding public key,
- 2) Compute a digital signature that may be verified by the corresponding public key,
- 3) Decrypt keys that were encrypted by the corresponding public key, or
- 4) Compute a shared secret during a key-agreement transaction.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.49****protected**

ciphered and /or digitally signed. Protection may be applied to xDLMS APDUs and/or to COSEM data.

**3.2.50****public key**

a cryptographic key used with a public key cryptographic algorithm that is uniquely associated with an entity and that may be made public. In an asymmetric (public) cryptosystem, the public key is associated with a private key. The public key may be known by anyone and, depending on the algorithm, may be used, for example, to:

- 1) Verify a digital signature that is signed by the corresponding private key,
- 2) Encrypt keys that can be decrypted using the corresponding private key, or
- 3) Compute a shared secret during a key-agreement transaction.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.51****public-key certificate**

a data structure that contains an entity's identifier(s), the entity's public key (including an indication of the associated set of domain parameters) and possibly other information, along with a signature on that data set that is generated by a trusted party, i.e. a certificate authority, thereby binding the public key to the included identifier(s).

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

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

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

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.53****Public Key Infrastructure****PKI**

a framework that is established to issue, maintain and revoke public key certificates.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.54****receiver <key-transport>**

the party that receives secret keying material via a key-transport transaction. Contrast with sender.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

**3.2.55****revoke a certificate**

to prematurely end the operational period of a certificate effective at a specific date and time

[SOURCE: NIST SP 800-32:2001]

**3.2.56****Root Certification Authority**

in a hierarchical Public Key Infrastructure, the Certification Authority whose public key serves as the most trusted datum (i.e., the beginning of trust paths) for a security domain

[SOURCE: NIST SP 800-32:2001]

**3.2.57****secret key**

a cryptographic key that is used with a secret key (symmetric) cryptographic algorithm that is uniquely associated with one or more entities and is not made public. The use of the term “secret” in this context does not imply a classification level, but rather implies the need to protect the key from disclosure

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.58****security services**

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

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.59****security strength****(also “bits of security”)**

a number associated with the amount of work (that is, the number of operations) that is required to break a cryptographic algorithm or system

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

**3.2.60****self-signed certificate**

a public key certificate whose digital signature may be verified by the public key contained within the certificate. The signature on a self-signed certificate protects the integrity of the data, but does not guarantee authenticity of the information. The trust of self-signed certificates is based on the secure procedures used to distribute them.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.61****sender <key-transport>**

the party that sends secret keying material to the receiver in a key-transport transaction. Contrast with receiver.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	33/614
-----------------------	------------	-----------------------	--------

**3.2.62****session key**

cryptographic key established for use for a relatively short period of time. In DLMS/COSEM the dedicated key is a session key.

**3.2.63****shared secret**

a secret value that has been computed using a key agreement scheme and is used as input to a key-derivation function/method

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.64****signature generation**

uses a digital signature algorithm and a private key to generate a digital signature on data

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.65****signature verification**

uses a digital signature algorithm and a public key to verify a digital signature on data

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.66****signed data**

data upon which a digital signature has been computed

**3.2.67****static symmetric key**

key that is intended for use for a relatively long period of time and is typically intended for use in many instances of a DLMS/COSEM Application Association

Note to entry: In DLMS/COSEM it is known as global key.

**3.2.68****static key**

a key that is intended for use for a relatively long period of time and is typically intended for use in many instances of a cryptographic key establishment scheme. Contrast with an ephemeral key.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.69****Subordinate Certification Authority**

in a hierarchical PKI, a Certification Authority (CA) whose certificate signature key is certified by another CA, and whose activities are constrained by that other CA

[SOURCE: NIST SP 800-32:2001]

**3.2.70****symmetric key**

a single cryptographic key that is used with a secret (symmetric) key algorithm

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.71****symmetric key algorithm**

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

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.72****trust anchor**

a public key and the name of a certification authority that is used to validate the first certificate in a sequence of certificates. The trust anchor public key is used to verify the signature on a certificate issued by a trust anchor certification authority. The security of the validation process depends upon the authenticity and integrity of the trust anchor. Trust anchors are often distributed as self-signed certificates.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.73****trusted party**

a trusted party is a party that is trusted by an entity to faithfully perform certain services for that entity. An entity could be a trusted party for itself.

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

**3.2.74****trusted third party**

a third party, such as a CA, that is trusted by its clients to perform certain services. (By contrast, in a key establishment transaction, the participants, parties U and V, are considered to be the first and second parties.)

[SOURCE: NIST SP 800-56A Rev. 2: 2013]

**3.2.75****X.509 certificate**

the X.509 public-key certificate or the X.509 attribute certificate, as defined by the ISO/ITU-T X.509 standard. Most commonly (including in this document), an X.509 certificate refers to the X.509 public-key certificate.

[SOURCE: NIST SP 800-57:2012, Part 1]

**3.2.76****X.509 public key certificate**

a digital certificate containing a public key for entity and a name for the entity, together with some other information that is rendered unforgeable by the digital signature of the certification authority that issued the certificate, encoded in the format defined in the ISO/ITU-T X.509 standard.

[SOURCE: NIST SP 800-57:2012, Part 1]

### 3.3 Definitions and abbreviations related to the Galois/Counter Mode

The source of the definitions 3.3.1 to 3.3.13 abbreviations and symbols in this subclause is NIST SP 800-38D:2007.

#### 3.3.1

##### **Additional Authenticated Data**

###### **AAD**

input data to the authenticated encryption function that is authenticated but not encrypted

#### 3.3.2

##### **authenticated decryption**

function of GCM in which the ciphertext is decrypted into the plaintext, and the authenticity of the ciphertext and the AAD are verified

#### 3.3.3

##### **authenticated encryption**

function of GCM in which the plaintext is encrypted into the ciphertext and an authentication tag is generated on the AAD and the ciphertext

#### 3.3.4

##### **authentication tag**

###### **Tag, T**

cryptographic checksum on data that is designed to reveal both accidental errors and the intentional modification of the data

#### 3.3.5

##### **block cipher**

parameterized family of permutations on bit strings of a fixed length; the parameter that determines the permutation is a bit string called the key

#### 3.3.6

##### **ciphertext**

encrypted form of the plaintext

#### 3.3.7

##### **fixed field**

in the deterministic construction of IVs, the field that identifies the device or context for the instance of the authenticated encryption function

#### 3.3.8

##### **fresh**

for a newly generated key, the property of being unequal to any previously used key

#### 3.3.9

##### **GCM**

Galois/Counter Mode

#### 3.3.10

##### **initialization Vector**

###### **IV**

nonce that is associated with an invocation of authenticated encryption on a particular plaintext and AAD

Note to entry: For the purposes of this standard, the invocation field is the invocation counter.

36/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
--------	------------	-----------------------	-----------------------

**3.3.11****invocation field**

in the deterministic construction of IVs, the field that identifies the sets of inputs to the authenticated encryption function in a particular device or context

**3.3.12****key**

parameter of the block cipher that determines the selection of the forward cipher function from the family of permutations

**3.3.13****plaintext****P**

input data to the authenticated encryption function that is both authenticated and encrypted

**3.3.14****security control byte****SC**

byte that provides information on the ciphering applied

**3.3.15****security header****SH**

concatenation of the security control byte *SC* and the invocation counter:  $SH = SC \parallel IC$ .

**3.4 Definitions and abbreviations related to Wi-SUN****3.4.1****border router node**

device that acts as the control point for multiple router devices across a large network

**3.4.2****leaf node**

device that does not provide any routing capability

**3.4.3****operating class**

with regulatory domain, reference to regionally allowable frequency bands

NOTE to entry: Regulatory Domains and frequency bands are defined in [FANSPEC].

**3.4.4****Personal Area Network (PAN)**

network area subservient to a border router node

**3.4.5****regulatory domain**

with operating class, reference to regionally allowable frequency bands

NOTE to entry: Regulatory Domains and frequency bands are defined in [FANSPEC]

**3.4.6****router/forwarding node**

device that manages messages between end nodes and the border router

### 3.5 General abbreviations

Abbreviation	Meaning
.cnf	.confirm service primitive
.ind	.indication service primitive
.req	.request service primitive
.res	.response service primitive
AA	Application Association
AAA	Authentication Authorization and Accounting
ACK	Acknowledgement
AARE	A-Associate Response – an APDU of the ACSE
AARQ	A-Associate Request – an APDU of the ACSE
ABP	Activation by Personalisation
ACPM	Association Control Protocol Machine
ACSE	Association Control Service Element
AE	Application Entity
AES	Advanced Encryption Standard
AL	Application Layer
AP	Application Process
APDU	Application Layer Protocol Data Unit
API	Application Programming Interface
ASE	Application Service Element
ASO	Application Service Object
ATM	Asynchronous Transfer Mode
A-XDR	Adapted Extended Data Representation
base_name	The short_name corresponding to the first attribute ("logical_name") of a COSEM object
BD	Block Data
BER	Basic Encoding Rules
BFE	Broadcast Frame Exchange
BN	Block Number
BNA	Block Number Acknowledged
BS	Bit string
BTS	Block Transfer Streaming
BTW	Block Transfer Window
CA	Certification Authority
CCA	Clear Channel Assessment
C/D	Compression and Decompression
CF	Control Function
CL	Connectionless
class_id	COSEM interface class identification code
CMP	Certificate Management Protocol. Refer to RFC 4210.
CO	Connection-oriented
CoAP	Constrained Application Protocol (as defined by RFC 7252)
CoAP BT	Constrained Application Protocol Block Transfer (as defined by RFC 7959)

Abbreviation	Meaning
CON	Confirmable
COSEM	Comprehensive Semantic Model for Energy Management
COSEM_on_IP	The TCP-UDP/IP based COSEM communication profile
CRC	Cyclic Redundancy Check
CRL	Certificate revocation list. Refer to RFC 5280.
CSAP	Client Service Access Point
CSMA-CA	Carrier Sense Multiple Access – Channel Access
CSR	Certificate Signing Request
DAG	Directed Acyclic Graph
DCE	Data Communication Equipment (communications interface or modem)
DCS	Data Collection System
DevAddr	A 32-bit non-unique identifier assigned to an end-device statically or dynamically after a Join Procedure (depending on the activation mode) (LPWAN)
DEVEUI	An IEEE EUI-64 used to identify the device during the Join Procedure
DFE	Directed Frame Exchange
DFE ULAD	Directed Frame Exchange Upper Layer Application Data
DIO	DODAG Information Object
DISC	Disconnect (a HDLC frame type)
DLMS	Device Language Message Specification
DM	Disconnected Mode (a HDLC frame type)
DODAG	Destination Oriented Directed Acyclic Graph See [RFC 6550].
DSA	Digital Signature Algorithm specified in FIPS PUB 186-4:2013
DSAP	Data Link Service Access Point
DSO	Energy Distribution System Operator
DTE	Data Terminal Equipment (computers, terminals or printers)
EAPOL	Extensible Authentication Protocol Over LAN
EAP-TLS	Extensible Authentication Protocol – Transport Layer Security
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman key agreement protocol
ECDSA	Elliptic Curve Digital Signature Algorithm specified in ANSI X9.62 and FIPS PUB 186-4:2013
ECP	Elliptic Curve Point
EDFE	Exended Directed Frame Exchange
ETX	Expected Transmission Count. Number of expected packet transmissions required for error free reception at destination.
EUI-64	64-bit Extended Unique Identifier
FAN	Field Area Network
FCS	Frame Check Sequence
FD	Fan Data [Link]
FDDI	Fibre Distributed Data Interface
FE	Field Element (in relation with public key algorithms)
FIPS	Federal Information Processing Standard
F/R	Fragmentation and Reassembly
FRMR	Frame Reject (a HDLC frame type)

Abbreviation	Meaning
FTP	File Transfer Protocol
Gr	A GBT APDU received
GAK	Global Authentication Key
GBEK	Global Broadcast Encryption Key
GBT	General Block Transfer
GCM	Galois/Counter Mode (GCM), an algorithm for authenticated encryption with associated data
GMAC	A specialization of GCM for generating a message authentication code (MAC) on data that is not encrypted
GMT	Greenwich Mean Time
Gr.X	A field of a GBT APDU received
Gs	A GBT APDU sent
Gs.X	A field of a GBT APDU sent
GSM	Global System for Mobile communications
GUA	Global Unicast Address
GUEK	Global Unicast Encryption Key
GW	Gateway
HCS	Header Check Sequence
HDLC	High-level Data Link Control
HES	Head End System, also known as Data Collection System NOTE The HES may be owned by the energy provider or the utility
HHU	Hand Held Unit
HLS	High Level Security (COSEM)
HMAC	Keyed-Hash Message Authentication Code specified in FIPS 198-1
HSM	Hardware Security Module
HTTP	Hypertext Transfer Protocol
I	Information (a HDLC frame type)
IANA	Internet Assigned Numbers Authority
IC	Interface Class
ICMP	Internet Control Message Protocol
IDevID	Initial Device Identifier. See [FANSPEC]
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
IV	Initialization Vector
KEK	Key Encrypting Key
LAN	Local Area Network
LB	Last Block
LDN	Logical Device Name
LLC	Logical Link Control (Sublayer)
LLS	Low Level Security
LNAP	Local Network Access Point
LPDU	LLC Protocol Data Unit

Abbreviation	Meaning
L-SAP	LLC sublayer Service Access Point
LSB	Least Significant Bit
LSDU	LLC Service Data Unit
m	mandatory, used in conjunction with attribute and method definitions
MAC	Medium Access Control (sublayer)
MAC	Message Authentication Code (cryptography)
MHDS	Multi Hop Delivery Service
MIB	Management Information Base
MPL	Multicast Protocol for Low-Power and Lossy Networks
MSAP	MAC sublayer Service Access Point (in the HDLC based profile, it is equal to the HDLC address)
MSB	Most Significant Bit
MSC	Message Sequence Chart
MSDU	MAC Service Data Unit
<b>MTU</b>	<b>Maximum Transmission Unit</b>
N(R)	Receive sequence Number
N(S)	Send sequence Number
NDM	Normal Disconnected Mode
NGW	Network Gateway
NIST	National Institute of Standards and Technology
NNAP	Neighbourhood Network Access Point
<b>NON</b>	<b>Non-confirmable</b>
NRM	Normal Response Mode
o	optional, used in conjunction with attribute and method definitions
OBIS	Object Identification System
OCSP	Online Certificate Status Protocol
OID	Object Identifier
OOB	Out of Band
OS	Octet string
OSI	Open System Interconnection
OTA	Over The Air
P/F	Poll/Final
PAN	Personal area network
PAN-IE	PAN Information Element
PAR	Positive Acknowledgement with Retransmission
PDU	Protocol data unit
PhL	Physical Layer
PIB	PAN Information Base
PHSDU	PH SDU
PKCS	Public Key Cryptography Standard, established by RSA Laboratories
PKI	Public Key Infrastructure
PLC	Power line carrier
PPP	Point-to-Point Protocol
PSDU	Physical layer Service Data Unit

## DLMS/COSEM Architecture and Protocols

<b>Abbreviation</b>	<b>Meaning</b>
PSTN	Public Switched Telephone Network
RA	Registration Authority
RG	Radio gateway
RLRE	A-Release Response – an APDU of the ACSE
RLRQ	A-Release Request – an APDU of the ACSE
RNG	Random Number Generator
RNR	Receive Not Ready (a HDLC frame type)
RPL	IPv6 Routing Protocol for Low-Power and Lossy Networks. See [RFC 6550].
RQ	Receive Queue
RR	Receive Ready (a HDLC frame type)
RSA	Algorithm developed by Rivest, Shamir and Adelman; specified in ANSI X9.31 and PKCS #1.
S	A sequence of blocks in the RQ or SQ
SAP	Service Access Point
SCHC	Static Context Header Compression and fragmentation, a generic framework
SDU	Service Data Unit
SHA	Secure Hash Algorithm; specified in FIPS PUB 180-4:2012
SNMP	Simple Network Management Protocol
SNRM	Set Normal Response Mode (a HDLC frame type)
SQ	Send Queue
SSAP	Server Service Access Point
STR	Streaming
SUP	Suplicant. See [IEEE 802.1x]
tbsCertificate	To be signed certificate
TCP	Transmission Control Protocol
TDEA	Triple Data Encryption Algorithm
TL	Transport Layer
TLS	Transport Layer Security
TPDU	Transport Layer Protocol Data Unit
TWA	Two Way Alternate
UA	Unnumbered Acknowledge (a HDLC frame type)
UDP	User Datagram Protocol
UI	Unnumbered Information (a HDLC frame type)
ULA	Unique Local (IPv6) Address
UNC	Unbalanced operation Normal response mode Class
<b>URI</b>	<b>Uniform Resource Identifier</b>
USS	Unnumbered Send Status
V(R)	Receive state Variable
V(S)	Send state Variable
VAA	Virtual Application Association
WPDU	Wrapper Protocol Data Unit
xDLMS ASE	Extended DLMS Application Service Element
Wi-Fi	Wireless Fidelity
See also list of abbreviations specific to a cryptographic algorithm in the relevant clauses.	

### 3.6 Symbols related to the Galois/Counter Mode

Symbol	Meaning
$A$	Additional Authenticated Data, AAD
$AK$	Authentication key, a parameter that is part of the AAD
$C$	Ciphertext
$EK$	Encryption key, i.e. the block cipher key
$IC$	Invocation counter, part of the initialization vector. See also invocation field.
$IV$	Initialization Vector
$\text{len}(X)$	The bit length of the bit string $X$ .
$\text{LEN}(X)$	The octet length of the octet string $X$ .
$P$	Plaintext
$SC$	Security Control Byte
$SH$	Security Header
$\text{Sys-}T$	System title
$T$	Authentication tag
$t$	The bit length of the authentication tag. NOTE This is the same as $\text{len}(T)$
$X \amalg Y$	Concatenation of two strings, $X$ and $Y$ .

### 3.7 Symbols related to the ECDSA algorithm

Symbol	Meaning
$d$	The ECDSA private key, which is an integer in the interval $[1, n - 1]$ .
$Q = (x_Q, y_Q)$	An ECDSA public key. The coordinates $x_Q$ and $y_Q$ are integers in the interval $[0, q - 1]$ , and $Q = dG$ .
$k$	The ECDSA per-message secret number, which is an integer in the interval $[1, n - 1]$ .
$r$	One component of an ECDSA digital signature. It is an integer in $[1, n - 1]$ . See the definition of $(r, s)$ .
$s$	One component of an ECDSA digital signature. It is an integer in $[1, n - 1]$ . See the definition of $(r, s)$ .
$(r, s)$	An ECDSA digital signature, where $r$ and $s$ are the digital signature components.
$M$	The message that is signed using the digital signature algorithm.
$\text{Hash}(M)$	The result of a hash computation (message digest or hash value) on message $M$ using an approved hash function.

### 3.8 Symbols related to the key agreement algorithms

Symbol	Meaning
$d_{e, U}, d_{e, V}$	Party U's and Party V's ephemeral private keys. These are integers in the range $[1, n-1]$ .
$d_{s, U}, d_{s, V}$	Party U's and Party V's static private keys. These are integers in the range $[1, n-1]$ .
$ID_U$	The identifier of Party U (the initiator)
$ID_V$	The identifier of Party V (the responder)
$Q_{e, U}, Q_{e, V}$	Party U's and Party V's ephemeral public keys. These are points on the elliptic curve defined by the domain parameters.
$Q_{s, U}, Q_{s, V}$	Party U's and Party V's static public keys. These are points on the elliptic curve defined by the domain parameters.
$U, V$	Represent the two parties in a (pair-wise) key establishment scheme.
$Z$	A shared secret (represented as a byte string) that is used to derive secret keying material using a key derivation method. Source: NIST SP 800-56A Rev. 2: 2013

### 3.9 Abbreviations related to the DLMS/COSEM M-Bus communication profile

Abbrev	Term	Standard domain
ACC	Access number field	M-Bus
ALA	Application Layer Address	M-Bus
CFG	Configuration byte	M-Bus
CI <sub>ELL</sub>	CI field introducing the extended link layer (wireless M-Bus)	M-Bus
CI Field	Control Information field	M-Bus
CI <sub>TL</sub>	CI field introducing the transport layer	M-Bus
DTSAP	Destination Transport Service Access Point	Telecontrol
ELL	Extended Link Layer	M-Bus
ELLA	Extended Link Layer Address	M-Bus
FIN (bit)	Final Bit	Telecontrol
FT1.2	Data Integrity Format class FT1.2	Telecontrol
FT3	Data Integrity Format Class FT3	Telecontrol
LLA	Link Layer Address	M-Bus
STS	Status byte	M-Bus
STSAP	Source Transport Service Access Point	Telecontrol
wM-Bus	Wireless M-Bus	M-Bus

## 4 Information exchange in DLMS/COSEM

### 4.1 General

This Clause 4 introduces the main concepts of information exchange in DLMS/COSEM.

The objective of DLMS/COSEM is to specify a standard for a business domain oriented interface object model for devices and systems, as well as services to access the objects. Communication profiles to transport the messages through various communication media are also specified.

The term "devices" is an abstraction; consequently "device" may be any type of device for which this abstraction is suitable.

The COSEM object model is specified in DLMS UA 1000-1 – Part 2, the "Blue Book". The COSEM objects provide a view of the functionality of devices through their communication interfaces.

This Technical report, the "Green Book" specifies the DLMS/COSEM application layer, lower protocol layers and communication profiles.

The key characteristics of data exchange using DLMS/COSEM are the following:

- devices can be accessed by various parties: clients and third parties;
- mechanisms to control access to the resources of the device are provided; these mechanisms are made available by the DLMS/COSEM AL and the COSEM objects ("Association SN / LN" object, "Security setup" object);
- security and privacy is ensured by applying cryptographical protection to xDLMS messages and to COSEM data;
- low overhead and efficiency is ensured by various mechanisms including selective access, compact encoding and compression;
- at a site, there may be single or multiple devices. In the case of multiple devices at a site, a single access point can be made available;
- data exchange may take place either remotely or locally. Depending on the capabilities of the device, local and remote data exchange may be performed simultaneously without interfering with each other;
- various communication media can be used on local networks (LN), neighbourhood networks (NN) and wide area networks (WAN).

The key element to ensure that the above requirements are met is the Application Association (AA) – determining the contexts of the data exchange – provided by the DLMS/COSEM AL. For details, see the relevant clauses below.

### 4.2 Communication model

DLMS/COSEM uses the concepts of the Open Systems Interconnection (OSI) model to model information exchange between **devices** and data collection systems.

**NOTE** Information in this context comprises xDLMS messages and COSEM data.

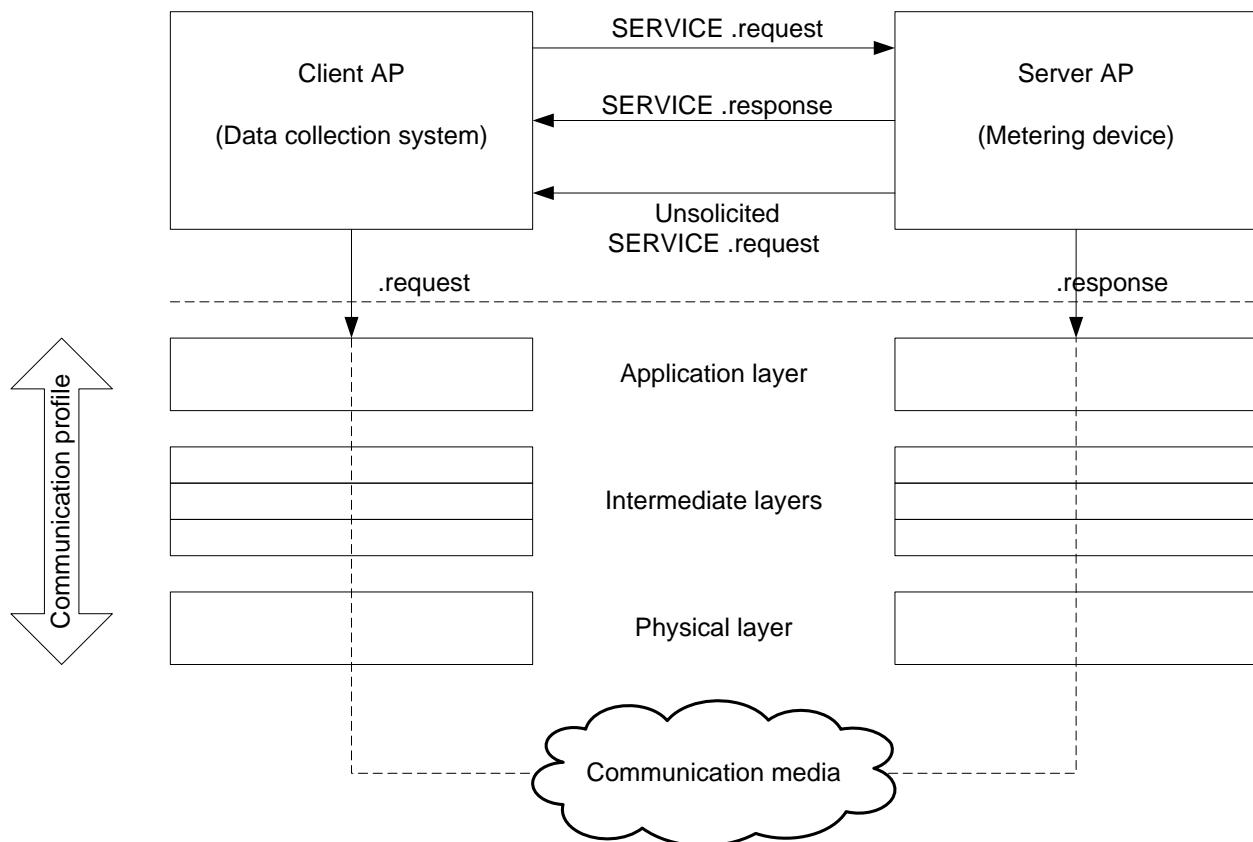
Concepts, names and terminology used below relate to the OSI reference model described in ISO/IEC 7498-1:1994. Their use is outlined in this clause and further developed in other clauses.

Application functions of devices and data collection systems are modelled by application processes (APs).

Communication between APs is modelled by communication between application entities (AEs). An AE represents the communication functions of an AP. There may be multiple sets of OSI communication functions in an AP, so a single AP may be represented by multiple AEs. However, each AE represents a single AP. An AE contains a set of communication capabilities called application service elements (ASEs). An ASE is a coherent set of integrated functions. These ASEs may be used independently or in combination. See also 9.1.2.

Data exchange between data collection systems and devices is based on the client/server model where data collection systems play the role of the client and devices play the role of the server. The client sends service requests to the server which sends service responses. In addition the server may initiate unsolicited service requests to inform the client about events or to send data on pre-configured conditions. See also 4.6.

In general, the client and the server APs are located in separate devices. Therefore, message exchange takes place via a protocol stack as shown in Figure 2.

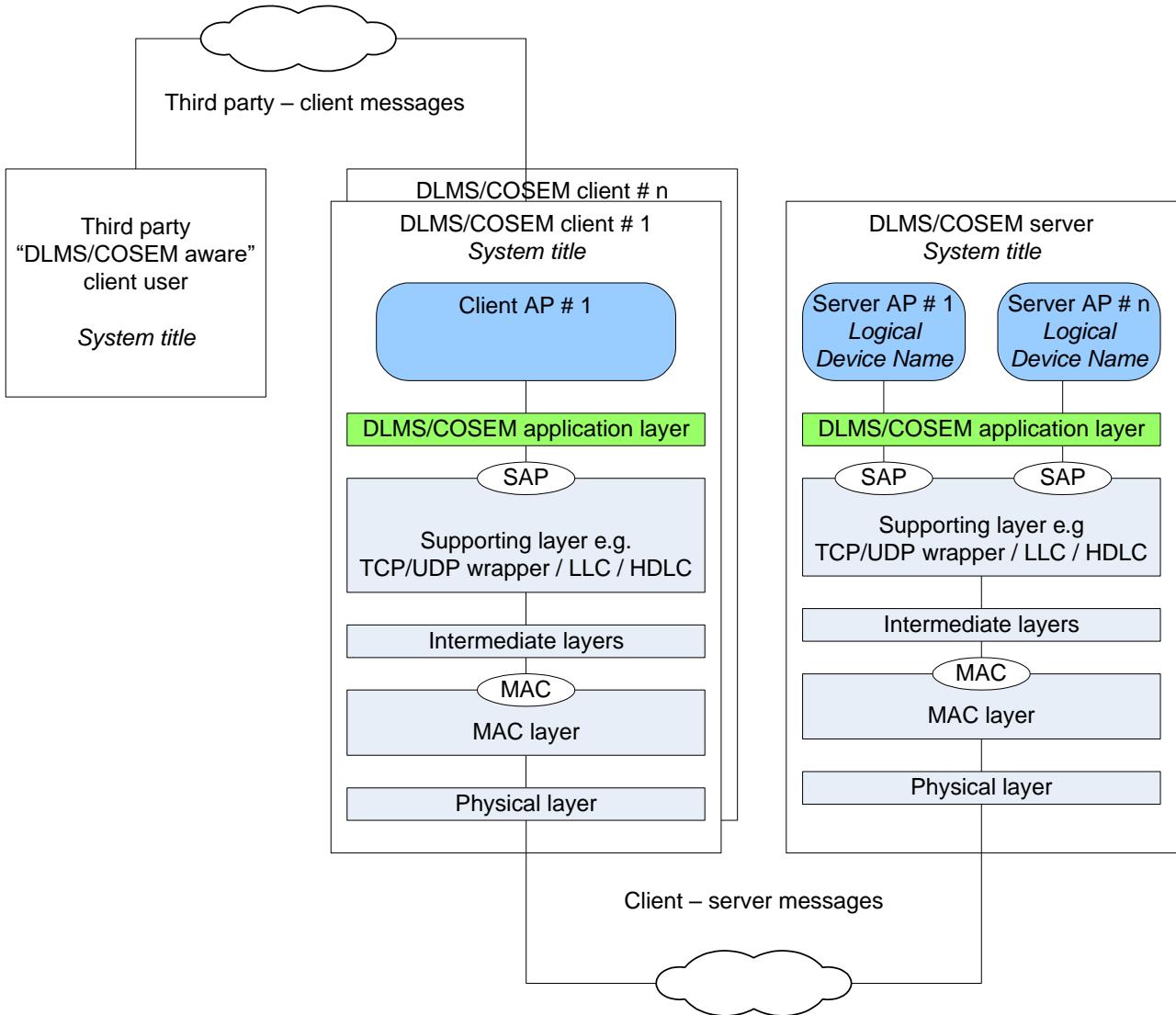


**Figure 2 – Client–server model and communication protocols**

## 4.3 Naming and addressing

### 4.3.1 General

Naming and addressing are important aspects in communication systems. A name identifies a communicating entity. An address identifies where that entity can be found. Names are mapped to addresses; this is known as the process of binding. Figure 3 shows the main elements of naming and addressing in DLMS/COSEM.



**Figure 3 – Naming and addressing in DLMS/COSEM**

#### 4.3.2 Naming

DLMS/COSEM entities, including clients, servers as well as third party systems shall be uniquely named by their *system title*. System titles shall be permanently assigned.

Server physical devices may host one or more logical devices (LDs). LDs shall be uniquely identified by their Logical Device Name (LDN). LDs hosted by the same physical device share the system title. System titles are specified in 4.3.4. Logical device names are specified in 4.3.5.

#### 4.3.3 Addressing

Each physical device shall have an appropriate address. It depends on the communication profile and may be a phone number, a MAC address, an IP network address, a CoAP URI, or a combination of these.

**NOTE** For example, in the case of the 3-layer, connection-oriented, HDLC based communication profile, the lower HDLC address is the MAC address.

Physical device addresses may be pre-configured or may be assigned during a registration process, which also involves binding between the addresses and the system titles.

Each DLMS client and each server – a COSEM logical device – is bound to a Service Access Point (SAP). The SAPs reside in the supporting layer of the DLMS/COSEM AL. Depending on the communication profile the SAP may be a TCP-UDP/IP wrapper address, a CoAP wrapper address, an

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	47/614
-----------------------	------------	-----------------------	--------

upper HDLC address, an LLC address etc. On the server side, this binding is modelled by the “SAP Assignment” IC; see **DLMS UA 1000-1 Part 2 Ed.15:2021**, 4.4.5.

The values of the SAPs on the client and the server side are specified in Table 1. The length of the SAPs depends on the communication profile.

**Table 1 – Client and server SAPs**

<b>Client SAPs</b>	
No-station	0x00
Client Management Process / CIASE <sup>1</sup>	0x01
Public Client	0x10
Open for client AP assignment	0x02 ...0x0F 0x11 and up
<b>Server SAPs</b>	
No-station / CIASE <sup>1</sup>	0x00
Management Logical Device	0x01
Reserved for future use	0x02...0x0F
Open for server SAP assignment	0x10 and up
All-station (Broadcast)	Communication profile specific
<sup>1</sup> In the case of the DLMS/COSEM S-FSK PLC profile, see 10.5.	
NOTE Depending on the supporting protocol layer, the SAPs may be represented on one or more bytes.	

#### 4.3.4 System title

The system title *Sys-T* shall uniquely identify each DLMS/COSEM entity. This may be a server, a client or a third party that can access servers via clients. The system title:

- shall be 8 octets long;
- shall be unique.

**The first three (most significant) octets** should hold the three-letter manufacturer ID<sup>1</sup>. This is the same as the first three octets of the Logical Device Name, see 4.3.5. The remaining 5 octets shall ensure uniqueness.

**NOTE** The system title can be derived for example from the last 12 digits of the manufacturing number, up to 999 999 999. This value converts to 0xE8D4A50FFF. Values above this, up to 0xFFFFFFFFFFF (decimal 1 099 511 627 775) can also be used, but these values cannot be mapped to the last 12 digits of the manufacturing number.

Project specific companion specifications may specify a different structure. In that case, the details should be specified by the naming authority designated for the project.

The use of the system title in cryptographic protection of xDLMS messages and COSEM data is further specified in 9.2.3 and 9.2.7.

**The client and server require knowledge of each others’ system titles before the cryptographic security algorithms can be used in a ciphered application context. The following options are available for the exchange of system titles:**

- during the communication media specific registration process.  
For example, when the S-FSK PLC profile is used, system titles are exchanged during the registration process using the CIASE protocol; see 10.5.5;

<sup>1</sup> Administered by the FLAG Association in co-operation with the DLMS UA.

- in all communication profiles, during AA establishment using the COSEM-OPEN service carried the AARQ / AARE APDU (see 9.3.2). If the system titles sent / received during AA establishment are not the same as the ones exchanged during the registration process, the AA shall be rejected;
- by writing the *client\_system\_title* attribute and by reading the *server\_system\_title* attribute of "Security setup" objects, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7.

In the case of broadcast communication the client can send its system title to all servers, but it has to retrieve the system title of each server one by one.

#### 4.3.5 Logical Device Name

Logical Device Name (LDN) shall be as specified in DLMS UA 1000-1 Part 2 Ed.15:2021, 4.1.8.2.

#### 4.3.6 Client user identification

The client user identification mechanism allows a server to distinguish between different users on the client side and to log their activities accessing the device. It is specified in DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.2. Naming of client users is outside the scope of this Technical Report.

### 4.4 Connection oriented operation

The DLMS/COSEM AL is connection oriented. See also 9.1.3.

A communication session consists of three phases, as it is shown in Figure 4:

- first, an application level connection, called Application Association (AA), is established between a client and a server AE; see also 9.1.3. Before initiating the establishment of an AA, the peer PhLs of the client and server side protocol stacks have to be connected. The intermediate layers may have to be connected or not. Each layer, which needs to be connected, may support one or more connections simultaneously;
- once the AA is established, message exchange can take place;
- at the end of the data exchange, the AA is released.



**Figure 4 – A complete communication session in the CO environment**

For the purposes of very simple devices, one-way communicating devices, and for multicasting and broadcasting pre-established AAs are also allowed. For such AAs the full communication session may include only the message exchange phase: it can be considered that the connection establishment phase has been already done somewhere in the past. Pre-established AAs cannot be released. See also 9.4.4.4.

### 4.5 Application associations

#### 4.5.1 General

Application Associations (AAs) are logical connections between a client and a server AE. AAs may be established on the request of a client using the services of the connection-oriented ACSE of the AL or may be pre-established. They may be confirmed or unconfirmed. See also 9.1.3.

NOTE 1 A pre-established AA can be considered to have been established in the past.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	49/614
-----------------------	------------	-----------------------	--------

NOTE 2 Servers cannot initiate the establishment of an AA.

A COSEM logical device may support one or more AAs, each with a different client. Each AA determines the contexts in which information exchange takes place.

A confirmed AA is proposed by the client and accepted by the server provided that:

- the user of the client is known by the server, see 4.3.6;
- the application context proposed by the client – see 4.5.2 – is acceptable for the server;
- the authentication mechanism proposed by the client – see 4.5.3 – is acceptable for the server and the authentication is successful;
- the elements of the xDLMS context – see 4.5.4 – can be successfully negotiated between the client and the server.

An unconfirmed AA is also proposed by a client with the assumption that the server will accept it. No negotiation takes place. Unconfirmed AAs are useful for sending broadcast messages from the client to servers.

AAs are modelled by COSEM “Association SN / LN” objects that hold the SAPs identifying the associated partners, the name of the *application context*, the name of the *authentication mechanism*, and the *xDLMS context*.

The “Association SN / LN” objects also determine a specific set of access rights to COSEM object attributes and methods and they point to (reference) a “Security setup” object that hold the elements of the security context. The access rights and the security context may be different in each AA. These objects are specified in DLMS UA 1000-1.

#### 4.5.2 Application context

The application context determines:

- the set of Application Service Elements (ASEs) present in the AL;
- the referencing style of COSEM object attributes and methods: short name (SN) referencing or logical name (LN) referencing. See also 9.1.4.3.1;
- the transfer syntax;
- whether ciphering is used or not.

Application contexts are identified by names, see 9.4.2.2.2.

#### 4.5.3 Authentication

In communication systems entity authentication is a fundamentally important security service. The goal of entity authentication is to establish whether the claimant of a certain identity is in fact who it claims to be. In order to achieve this goal, there should be a pre-existing relation which links the entity to a secret.

In DLMS/COSEM, authentication takes place during AA establishment.

In confirmed AAs either the client (unilateral authentication) or both the client and the server (mutual authentication) can authenticate itself.

In an unconfirmed AA, only the client can authenticate itself.

In pre-established AAs, authentication of the communicating partners is not available.

Once the AA is established, COSEM object attributes and methods can be accessed using xDLMS services subject to the prevailing security context and access rights in the given AA.

The COSEM authentication mechanisms are specified in 9.2.2.2.2. The authentication mechanisms are identified by names, see 9.4.2.2.3.

#### 4.5.4 xDLMS context

The xDLMS context determines the set of xDLMS services and capabilities that can be used in a given AA. See 9.1.4.

50/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
--------	------------	-----------------------	-----------------------

#### 4.5.5 Security context

The security context is relevant when the application context stipulates ciphering. It comprises the security suite, the security policy, the security keys and other security material. See also 9.2.2.3. It is managed by “Security setup” objects.

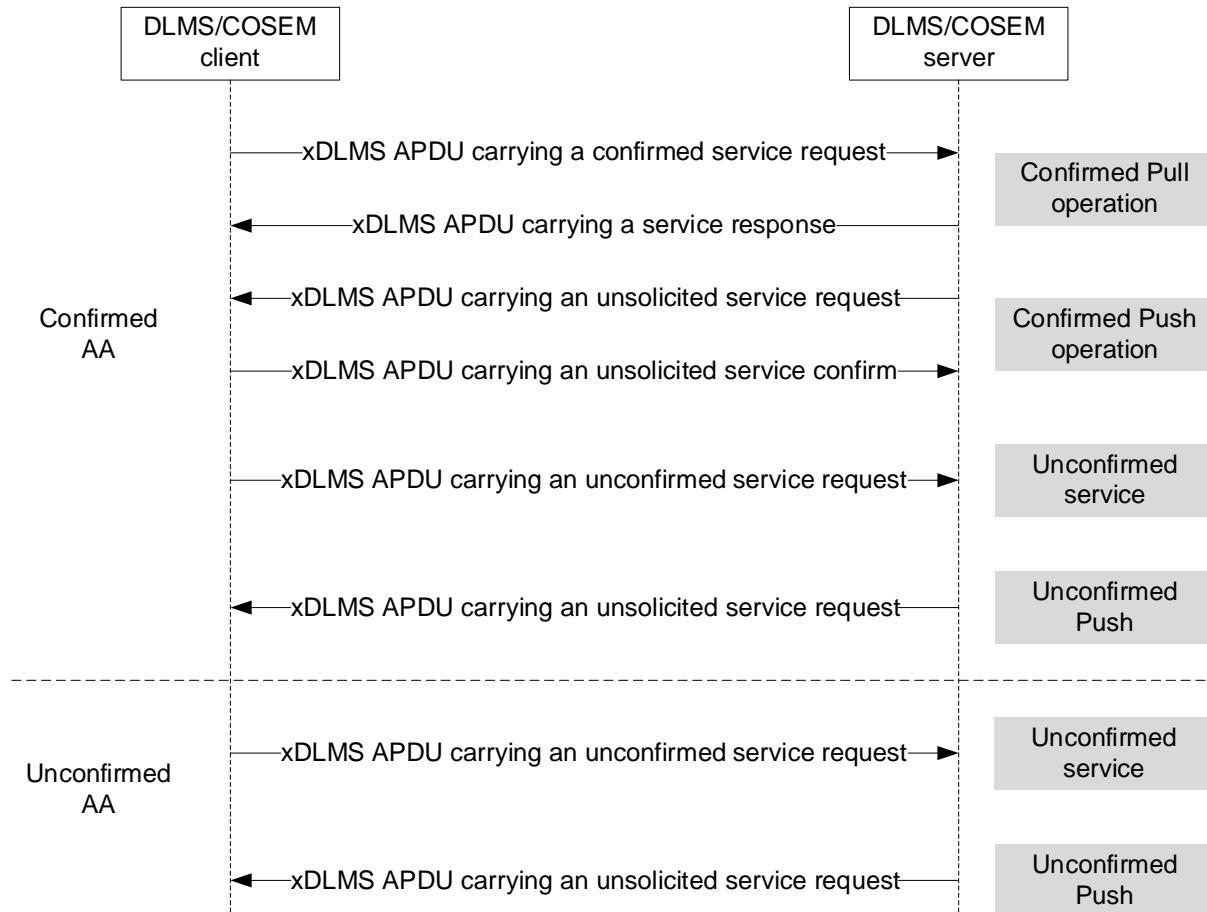
#### 4.5.6 Access rights

Access rights determine the rights of the client(s) to access COSEM object attributes and methods within an AA. The set of access rights depend on the role of the client and is pre-configured in the server. See also 9.2.2.4.

**NOTE** The roles and the related access rights are subject to project specific companion specifications. Examples for roles are device reader, device service / communication service / energy provider, manufacturer, end user etc.

### 4.6 Messaging patterns

The messaging patterns available between a DLMS client and server are shown in Figure 5.



**Figure 5 – DLMS/COSEM messaging patterns**

In confirmed AAs:

- the client can send confirmed service requests and the server responds: *pull operation*;
- the client can send unconfirmed service requests. The server does not respond;
- the server can send unsolicited service requests to the client: *push operation*.

**NOTE** The unsolicited services may be InformationReport (with SN referencing), EventNotification (with LN referencing) or DataNotification (used both with SN and LN referencing).

In unconfirmed AAs:

- only the client can initiate service requests and only unconfirmed ones. The server cannot respond and it cannot initiate service requests.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	51/614
-----------------------	------------	-----------------------	--------

#### 4.7 Data exchange between third parties and DLMS servers

Third parties – that are outside the DLMS client-server relationship – may also exchange information with servers, using a client as a broker. To support end-to-end security, such third parties shall be “DLMS/COSEM aware” meaning that they shall be able to send messages to the client that contain properly formatted xDLMS APDUs carrying properly formatted COSEM data, and that they shall be able to process messages received from the server via the client. See also 9.2.2.5, Figure 91.

Messages from the server to the third party may be solicited or unsolicited.

#### 4.8 Communication profiles

Communication profiles specify how the DLMS/COSEM AL and the COSEM data model modelling the Application Process (AP) are supported by the lower, communication media specific protocol layers.

Communication profiles comprise a number of protocol layers. Each layer has a distinct task and provides services to its upper layer and uses services of its supporting protocol layer(s). The client and server COSEM APs use the services of the highest protocol layer, that of the DLMS/COSEM AL. This is the only protocol layer containing COSEM specific element(s): the xDLMS ASE; see 9.1.4. It may be supported by any layer capable of providing the services required by the DLMS/COSEM AL. The number and type of lower layers depend on the communication media used.

A given set of protocol layers with the DLMS/COSEM AL and the COSEM object model on top constitutes a particular DLMS/COSEM communication profile. Each profile is characterized by the protocol layers included and their parameters.

Figure 6 shows a generic DLMS/COSEM communication profile, including:

- the COSEM object model modelling the Application Process. For each communication media, media-specific setup interface classes are specified;
- the DLMS/COSEM application layer;
- the DLMS/COSEM TCP/UDP based transport layer, present in internet capable profiles;
- **the DLMS/COSEM CoAP based transport layer, present in internet capable profiles;**
- the convergence layers that bind the MAC layer to the DLMS/COSEM AL either directly or through the DLMS/COSEM TCP-UDP based transport layer or through the CoAP based DLMS/COSEM transport layer;
- the media specific physical and MAC layers; and
- the connection managers, when applicable.

A single physical device may support more than one communication profile to allow data exchange using various communication media. In such cases it is the task of the client side AP to decide which communication profile should be used.

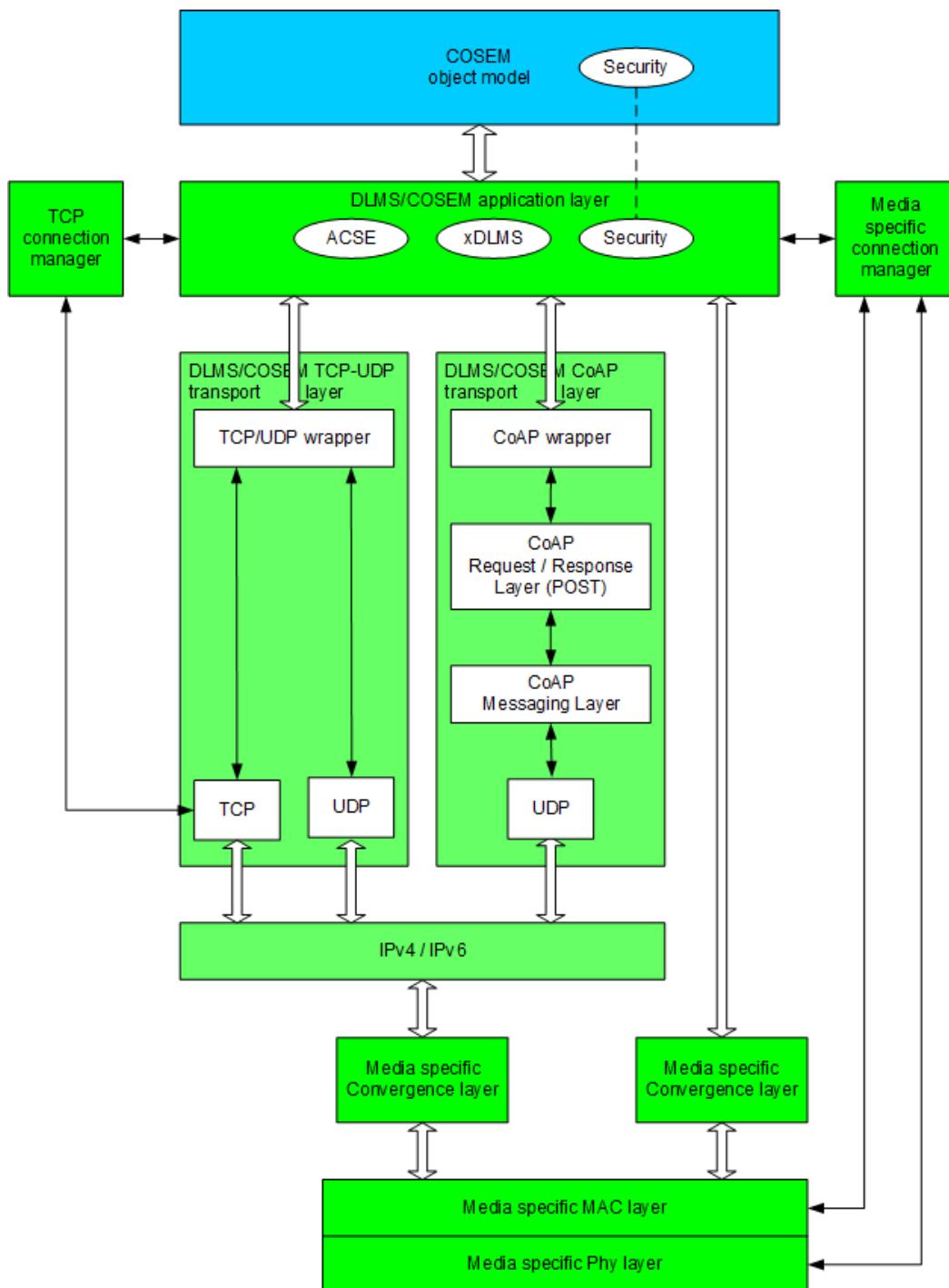


Figure 6 – DLMS/COSEM generic communication profile

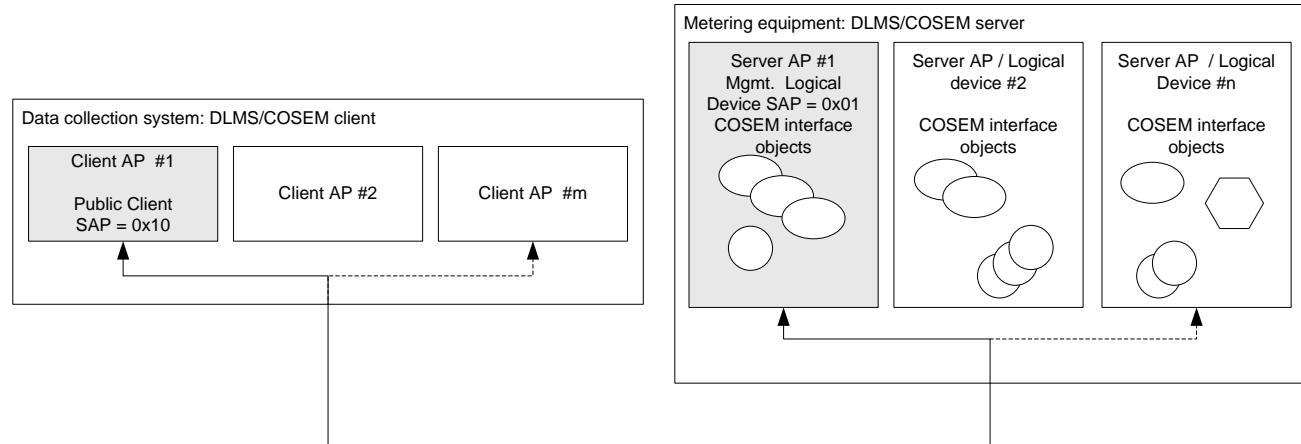
Communication profiles are specified in Clause 10:

- the elements to be specified in a communication profile are specified in 10.1;
- The 3-layer, connection-oriented, HDLC based communication profile, specified in 10.2;
- The TCP-UDP/IP based communication profiles (COSEM\_on\_IP), specified in 10.3;
- the CoAP based communication profile (DLMS/COSEM\_on\_CoAP) specified in 10.4
- The S-FSK PLC profile, specified in 10.5;
- The wired and wireless M-Bus profile, specified in 10.6;
- the LPWAN profile that can be used over LoRaWAN networks, specified in 10.8;
- the Wi-SUN profile specified in 10.9.
- the gateway protocol specified in 10.10;

#### 4.9 Model of a DLMS/COSEM system

Figure 7 shows a model of a DLMS/COSEM system.

Equipment is modelled as a set of logical devices, hosted in a single physical device. Each logical device represents a server AP and models a subset of the functionality of the equipment as these are seen through its communication interfaces. The various functions are modelled using COSEM objects.



**Figure 7 – Model of a DLMS/COSEM system**

Data collection systems are modelled as a set of client APs that may be hosted by one or several physical devices. Each client AP may have different roles and access rights, granted by the equipment.

The Public Client and the Management Logical Device APs have a special role and they shall be always present.

See more in DLMS UA 1000-1 Part 2 Ed.15:2021, 4.1.7 and 4.1.8.

#### 4.10 Model of DLMS servers

Figure 8 shows the model of two DLMS servers as an example. One of them uses a 3-layer, CO, HDLC based communication profile, and the other one uses a TCP-UDP/IP based communication profile.

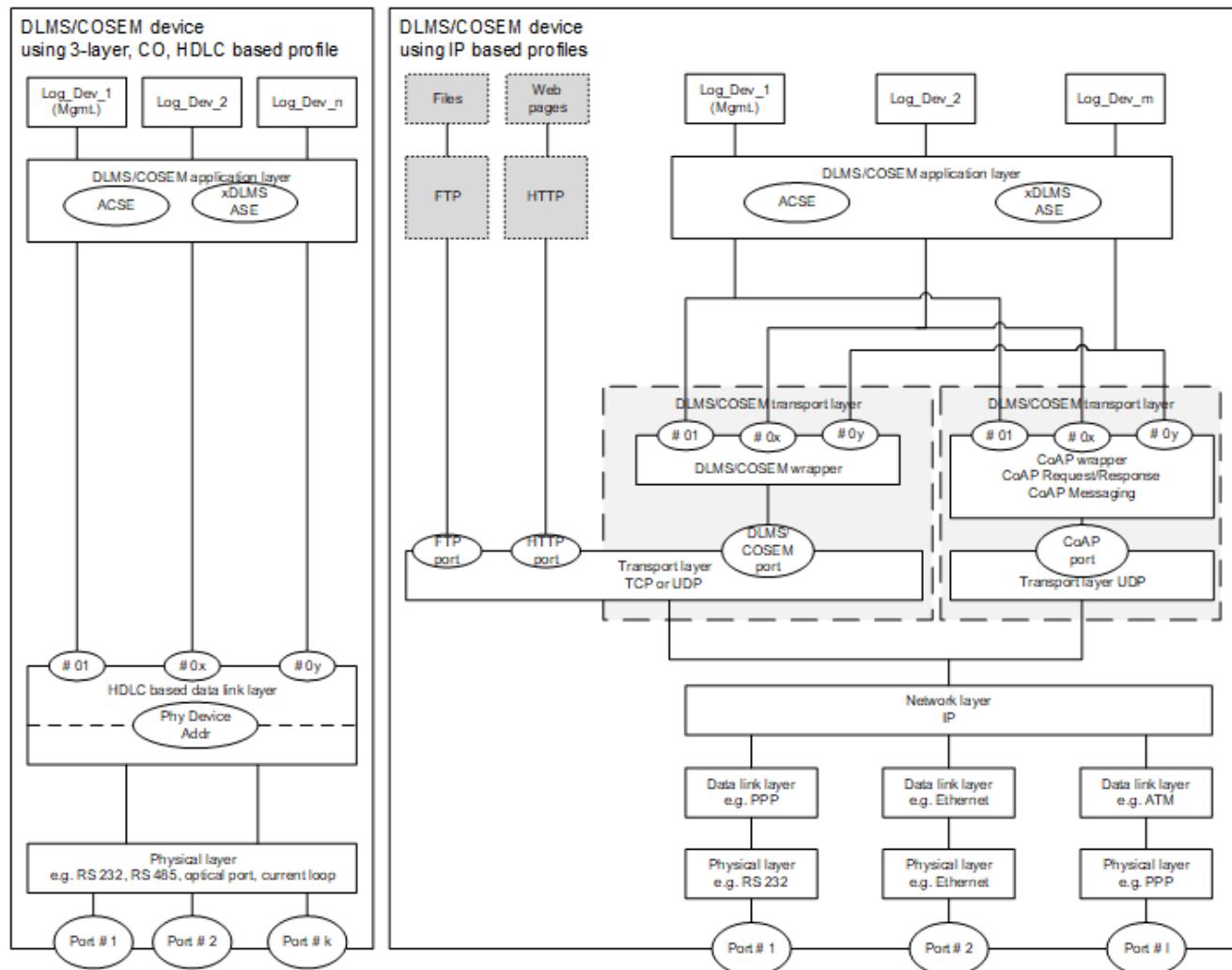
The equipment on the left hand side comprises "n" logical devices and supports the 3-layer, CO, HDLC based communication profile.

The DLMS/COSEM AL is supported by the HDLC based data link layer. Its main role is to provide a reliable data transfer between the peer layers. It also provides addressing of the logical devices in such a way, that each logical device is bound to a single HDLC address. The Management Logical Device is always bound to the address 0x01. To allow creating a local network so that several devices at a given site can be reached through a single access point, another address, the physical address is also provided by the data link layer. The logical device addresses are referred to as upper HDLC addresses, while the physical device address is referred to as a lower HDLC address. See also 8.4.2.

The PhL supporting the data link layer provides serial bit transmission between physical devices hosting the client and server applications. This allows using various interfaces, like RS 232, RS 485, 20 mA current loop, etc. to transfer data locally through PSTN and GSM networks etc.

The equipment on the right hand side comprises "m" logical devices.

The DLMS/COSEM AL is supported by the DLMS/COSEM TL, comprising the internet TCP or UDP layer and a wrapper. The main role of the wrapper is to adapt the OSI-style service set, provided by the DLMS/COSEM TL to and from TCP and UDP function calls. It also provides addressing for the logical devices, binding them to a SAP called wrapper port. The Management Logical Device is always bound to wrapper port 0x01. Finally, the wrapper provides information about the length of the APDUs transmitted, to help the peer to recognise the end of the APDU. This is necessary due the streaming nature of TCP.



**Figure 8 – DLMS server model**

Figure 8 shows the model of two DLMS servers as an example. One of them uses a 3-layer, CO, HDLC based communication profile, the other uses a TCP-UDP/IP based communication profile or a CoAP based communication profile or both.

The server on the left-hand side comprises “n” logical devices and supports the 3-layer, CO, HDLC based communication profile.

The DLMS/COSEM AL is supported by the HDLC based data link layer. Its main role is to provide a reliable data transfer between the peer layers. It also provides addressing of the logical devices in such a way, that each logical device is bound to a single HDLC address. The Management Logical Device is always bound to the address 0x01. A physical address is also provided by the data link layer to allow the creation of a local network so that several physical devices at a given site can be reached through a single access point. The logical device addresses are referred to as upper HDLC addresses, while the physical device address is referred to as a lower HDLC address. See also 8.4.2

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	55/614
-----------------------	------------	-----------------------	--------

The PhL supporting the data link layer provides serial bit transmission between physical devices hosting the client and server applications. This allows using various interfaces, like RS 232, RS 485, 20 mA current loop, etc. to transfer data locally through PSTN and GSM networks etc.

The server on the right-hand side comprises "m" logical devices.

The DLMS/COSEM AL is supported by the DLMS/COSEM TL. This comprises the internet TCP or UDP layer or CoAP layer and UDP layer and a wrapper. There are two choices of wrapper, one supporting TCP/IP directly through a dedicated TCP or UDP port number allocated for the DLMS/COSEM AL, the other is to support the use of CoAP over UDP via a CoAP port.

The main role of the wrapper is to adapt the OSI-style service set, provided by the DLMS/COSEM TL to and from TCP and UDP or CoAP function calls. It also provides addressing for the logical devices, binding them to a SAP called wrapper port. The Management Logical Device is always bound to wrapper port 0x01. In the case of TCP, the wrapper provides information about the length of the APDUs transmitted, to help the peer to recognise the end of the APDU. This is necessary due the streaming nature of TCP.

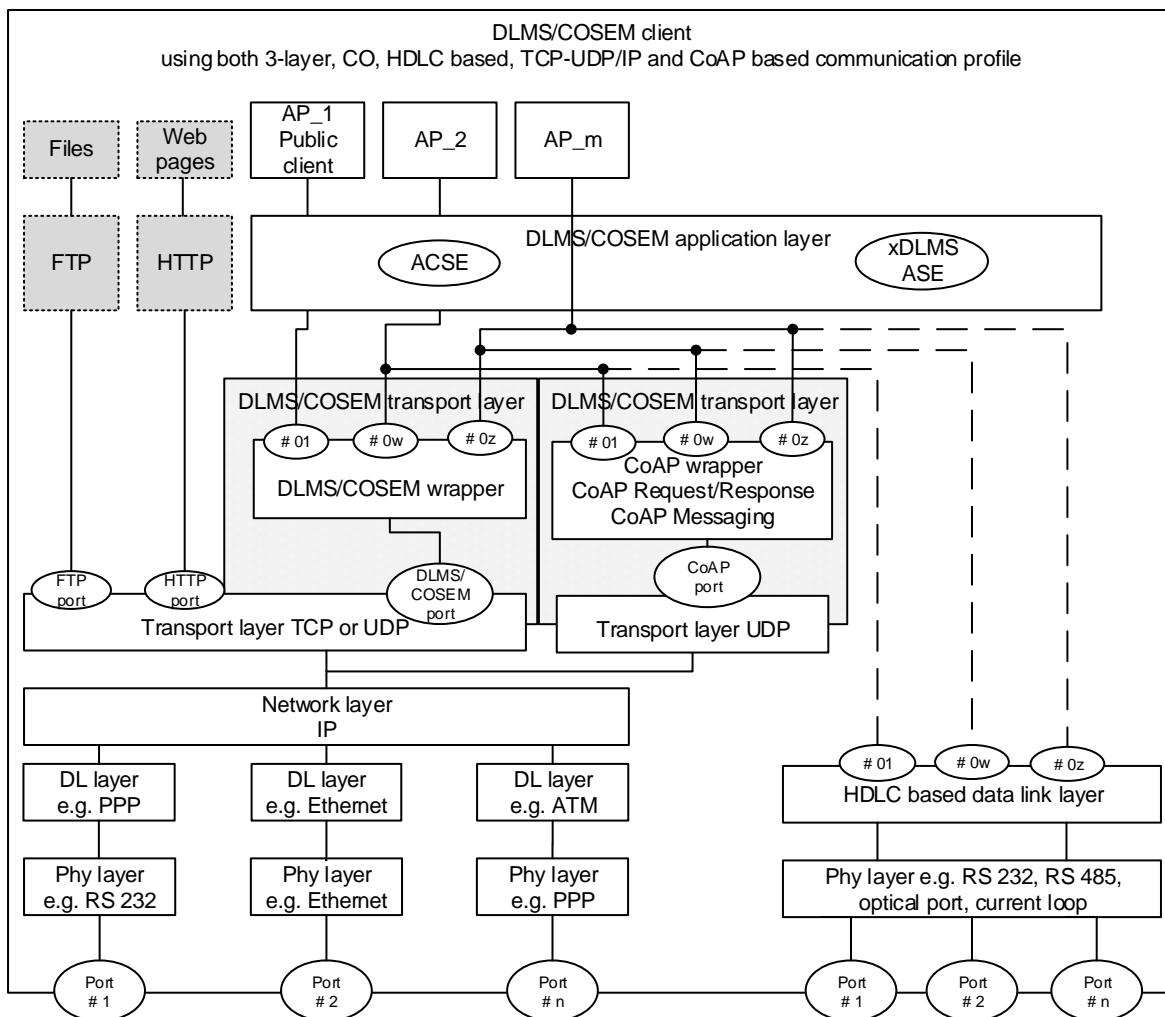
In the TCP-UDP/IP based profiles the DLMS/COSEM AL is bound to a TCP or UDP port number through the wrapper which is used for the DLMS/COSEM application. The presence of the TCP and UDP layers allows incorporating other internet applications, like FTP or HTTP, bound to their standard ports respectively.

The TCP and UDP layers are supported by the IP layer, which is in turn may be supported by any set of lower layers depending on the communication media to be used (for example Ethernet, PPP, IEEE 802, or IP-capable PLC lower layers, etc.).

It is possible to implement several protocol stacks in a single server with the common DLMS/COSEM AL being supported by distinct sets of lower layers. This allows the server to exchange data via various communication media with clients in different AAs. Such a structure would be similar to the structure of a DLMS client show in Figure 9.

#### 4.11 Model of a DLMS client

Figure 9 shows the model of a DLMS client as an example.



**Figure 9 – Model of a DLMS client using multiple protocol stacks**

The model of the client is very similar to the model of the servers:

- in this particular model, the DLMS/COSEM AL is supported either by the HDLC based data link layer or the IP-based DLMS/COSEM TLs, meaning that the AL uses the services of one or the other as determined by the APs. In other words, the APDUs are received from or sent through the appropriate supporting protocol layer, which in turn use the services of its supporting protocol layer respectively;
- unlike on the server side, the addressing provided by the HDLC layer has a single level only, that of the Service Access Points (SAP) of each Application Process (AP).

Client APs and server APs are identified by their SAPs, therefore, an AA between a client and a server side AP can be identified by a pair of client and server SAPs.

The DLMS/COSEM AL may be capable of supporting one or more AAs simultaneously. Likewise, lower layers may be capable of supporting more than one connection with their peer layers. This allows data exchange between clients and servers simultaneously via different ports and communication media.

#### 4.12 Interoperability and interconnectivity in DLMS/COSEM

In the DLMS/COSEM environment, interoperability and interconnectivity is defined between client and server AEs. A client and a server AE must be interoperable and interconnectable to ensure data exchange between the two systems.

Using the COSEM object model to model devices of all kinds , over all communication media ensures *semantic interoperability*, i.e. an unambiguous, shared meaning between clients and servers using different communication media. The semantic elements are the COSEM objects, their logical name i.e. the OBIS code, the definition of their attributes and methods and the data types that can be used.

Using the DLMS/COSEM AL over all communication media ensures *syntactic interoperability*, which is a pre-requisite of *semantic interoperability*. Syntactic interoperability comprises the ability to establish AAs between clients and server using various application contexts, authentication mechanisms, xDLMS contexts and security contexts as well as the standard structure and encoding of all messages exchanged.

*Interconnectivity* is a protocol level notion: in order to be able to exchange messages, the client and the server AEs should be **interconnectable** and **interconnected**.

Before the two AEs can establish an AA, they must be *interconnected*. The two AEs are interconnected, if each peer protocol layer of both sides, which needs to be connected, is connected. In order to be interconnected, the client and server AEs should be interconnectable and shall establish the required connections. Two AEs are *interconnectable* if they use the same communication profile.

With this, interconnectivity in DLMS/COSEM is ensured by the ability of the DLMS/COSEM AE to establish a connection between all peer layers, which need to be connected.

#### 4.13 Ensuring interconnectivity: the protocol identification service

In DLMS/COSEM, AA establishment is always initiated by the client AE. However, in some cases, it may not have knowledge about the protocol stack used by an unknown server device (for example when the server has initiated the physical connection establishment). In such cases, the client AE needs to obtain information about the protocol stack implemented in the server.

A specific, application level service is available for this purpose: the protocol identification service. It is an optional application level service, allowing the client AE to obtain information – after establishing a physical connection – about the protocol stack implemented in the server. The protocol identification service, specified in 5.3.3.3, uses directly the data transfer services (PH-DATA.request/.indication) of the PhL; it bypasses the other protocol layers. It is recommended to support it in all communication profiles that have access to the PhL.

#### 4.14 System integration and installation

System integration is supported by DLMS/COSEM in a number of ways.

A possible process is described here.

As shown in Figure 7, the presence of a Public Client (bound to address 0x10 in any profile) is mandatory in each client system. Its main role is to reveal the structure of an unknown – for example newly installed – device. This takes place within a mandatory AA between the Public Client and the Management Logical Device, with no security precautions. Once the structure is known, data can be accessed with using the proper authentication mechanisms and cryptographic protection of the xDLMS messages and COSEM data.

When a new device is installed in the system, it may generate an event report to the client. Once this is detected, the client can retrieve the internal structure of the device, and then send the necessary configuration information (for example tariff schedules and installation specific parameters) to the device. With this, the device is ready to use.

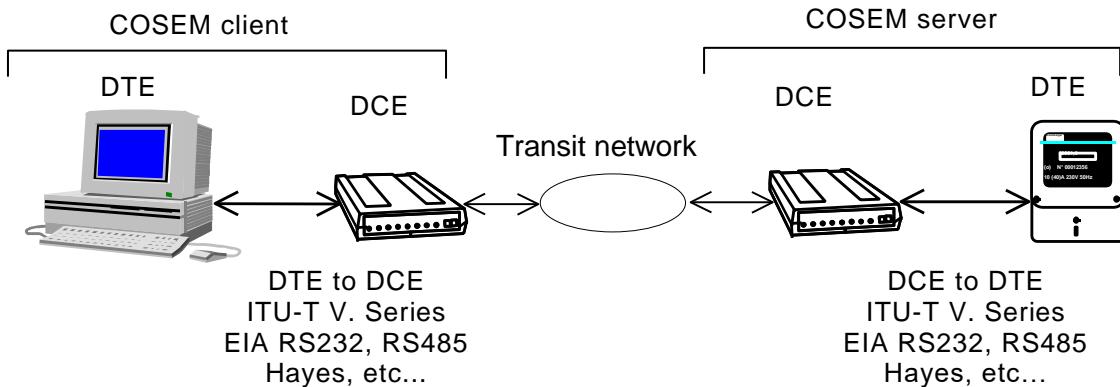
System integration is also facilitated by the availability of the DLMS/COSEM conformance testing, described in the Yellow Book, DLMS UA 1001-1. With this, correct implementation of the specification in equipment can be tested and certified.

## 5 Physical layer services and procedures for connection-oriented asynchronous data exchange

**NOTE** The physical layer specified here is described primarily for use in the 3-layer, CO, HDLC based communication profile. The physical layer of the S-FSK PLC communication profile is described in 10.5.4.2. The physical layer to be used in the TCP-UDP/IP based communication profile is out of the Scope of this Technical Report.

### 5.1 Overview

From the external point of view, the physical layer (PhL) provides the interface between the Data Terminal Equipment, DTE, and the Data Communication Equipment, DCE, see Figure 11. Figure 10 shows a typical configuration for data exchange through a wide area network, for example the PSTN.



**Figure 10 – Typical PSTN configuration**

From the physical connection point of view, all communications involve two sets of equipment represented by the terms caller system and called system. The caller system is the system that decides to initiate a communication with a remote system known as the called system; these denominations remain valid throughout the duration of the communication. A communication is broken down into a certain number of transactions. Each transaction is represented by a transmission from the transmitter to the receiver. During the sequence of transactions, the caller and called systems take turns to act as transmitter and receiver.

From the data link point of view, the DCS normally acts as a master (primary station), taking the initiative and controlling the data flow. The equipment is the slave (secondary station), responding to the primary station.

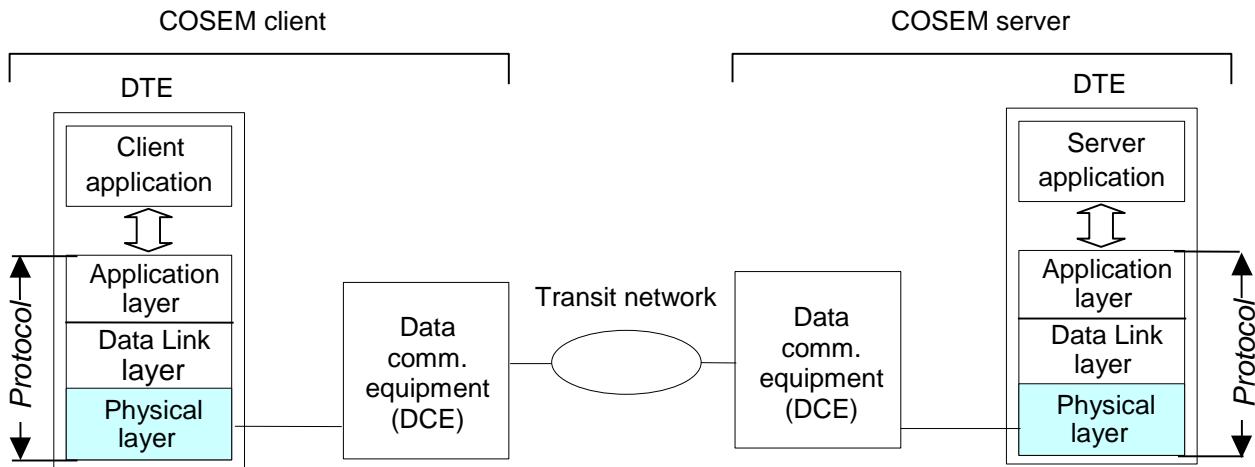
From the application point of view, the DCS normally acts as a client asking for services, and the equipment acts as a server delivering the requested services.

The situation involving a caller client and a called server is undoubtedly the most frequent case, but a communication based on a caller server and a called client is also possible, in particular to report the occurrence of an urgent alarm.

For the purposes of local data exchange, two DTEs can be directly connected using appropriate connections. To allow using a wide variety of media, this Technical Report does not specify the PhL signals and their characteristics. However, the following assumptions are made:

- the communication is point to point or point to multipoint;
- at least half-duplex connections are possible;
- asynchronous transmission with 1 start bit, 8 data bits, no parity and 1 stop bit (8N1).

From the internal point of view, the PhL is the lowest layer in the protocol stack.

**Figure 11 – The location of the physical layer**

In the following, the services of the PhL towards its peer layer(s) and the upper layers, as well as the protocol of the PhL are defined.

## 5.2 Service specification

### 5.2.1 List of services

ITU-T X.211 defines a set of capabilities to be made available by the PhL over the physical media. These capabilities are available via services, as follows:

- Connection establishment/release related services: PH-CONNECT, PH-ABORT;
- Data transfer services: PH-DATA;
- Layer management services.

Layer management services are used by or provided for the layer management process, which is part of the AP. Some examples are given below:

- PH-INITIALIZE.request / PH-INITIALIZE.confirm;
- PH-GET\_VALUE.request / PH-GET\_VALUE.confirm;
- PH-SET\_VALUE.request / PH-SET\_VALUE.confirm;
- PH-LM\_EVENT.indication.

As these services are of local importance only, their definition is not within the Scope of this Technical Report.

### 5.2.2 Use of the physical layer services

Figure 12 shows how different service users use the service primitives of the PhL. As it can be seen, the physical connection establishment/release services are used by and provided for the physical connection manager AP, and not the data link layer. The reasons for this are explained in 5.3.3.1.

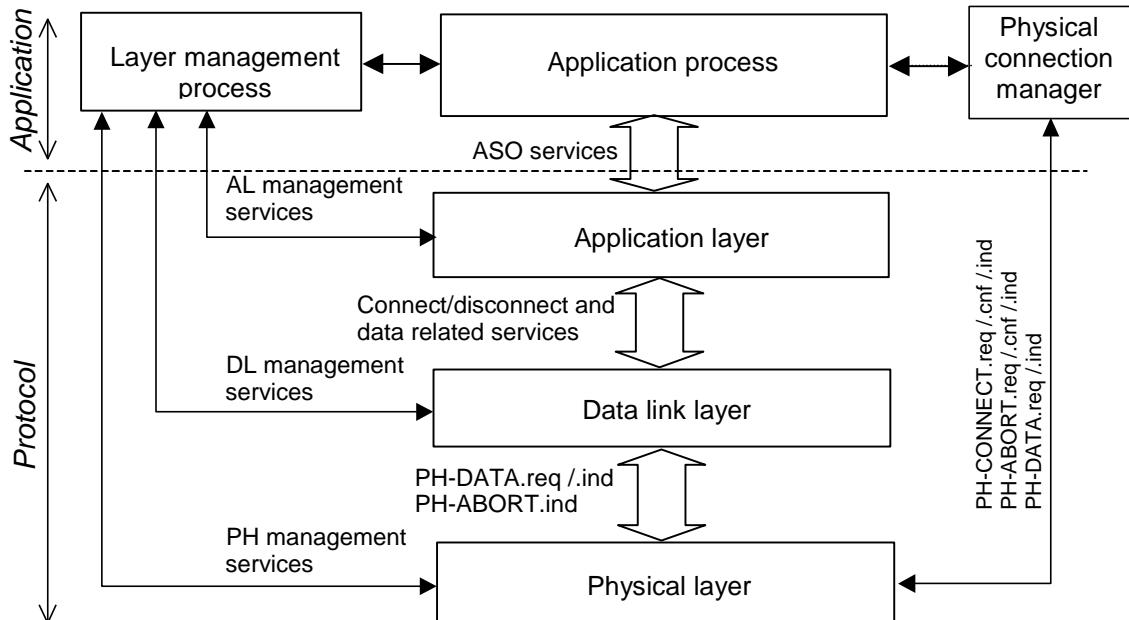


Figure 12 – Protocol layer services of the COSEM 3-layer connection-oriented profile

### 5.2.3 Service definitions

#### 5.2.3.1 The PH-CONNECT service

##### 5.2.3.1.1 PH-CONNECT.request

###### 5.2.3.1.1.1 Function

This primitive is the service request primitive of the connection establishment service.

###### 5.2.3.1.1.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
PH-CONNECT.request ( 
    PhConnType,
    PhConnReqParams
)
```

Where:

- PhConnType specifies the type of connection requested, for example direct connection, PSTN modem connection, etc. This Technical Report does not specify data type(s) and/or value(s) for this parameter, because this is a local issue only.
- The structure and the contents of the PhConnReqParams parameter depend on the value of the PhConnType parameter. For example, in the case of a PSTN connection it includes the phone number of the remote station, etc. As – similarly to the PhConnType parameter – the PhConnReqParams parameter contains implementation dependent data, data types / values for this parameter are not specified in this Technical Report.

###### 5.2.3.1.1.3 Use

In the DLMS/COSEM environment, the user of the PH-CONNECT.request primitive is the physical connection manager AP. It is used for the establishment of a physical connection. The receipt of this primitive causes the PhL entity to perform the required actions – for example to dial the specified phone number – to establish a physical connection with the peer PhL entity. An example of these actions in the case of an intelligent Hayes modem is given in 5.4.

### 5.2.3.1.2 PH-CONNECT.indication

#### 5.2.3.1.2.1 Function

This primitive is the service indication primitive of the connection establishment service.

#### 5.2.3.1.2.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

**PH-CONNECT.indication ( )**

#### 5.2.3.1.2.3 Use

The PH-CONNECT.indication is generated by the PhL entity primitive to indicate to the service user entity that a remote device requests a physical connection be established.

### 5.2.3.1.3 PH-CONNECT.confirm

#### 5.2.3.1.3.1 Function

This primitive is the service confirm primitive of the connection establishment service.

#### 5.2.3.1.3.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

**PH-CONNECT.confirm (**  
Result,  
PhConnCnfParams  
**)**

Where:

- Result indicates whether the attempt to set up a physical connection was successful or not.
- The structure and the value of the PhConnCnfParams parameter depend on the physical connection type of the corresponding PH-CONNECT.request primitive, which is actually being confirmed. For example, in the case of a PSTN connection it may include parameters of the established connection (V.22, baud-rate, etc.). Data types and values for this parameter are not specified in this Technical Report.

#### 5.2.3.1.3.3 Use

The PH-CONNECT.confirm primitive is used by the PhL entity to convey the results of the associated PH-CONNECT.request. If the connection could not be established due to a local error – for example the phone line is not available – it is locally generated.

### 5.2.3.2 The PH-DATA service

#### 5.2.3.2.1 PH-DATA.request

##### 5.2.3.2.1.1 Function

This primitive is the service request primitive of the data transfer service.

##### 5.2.3.2.1.2 Semantics of the service primitive

The semantics of this primitive is as follows:

**PH-DATA.request (**  
Data  
**)**

The Data parameter carries the byte to be transmitted by the PH layer entity.

##### 5.2.3.2.1.3 Use

The PH-DATA.request primitive is invoked by the service user entity to request sending a data byte to one or several remote PhL entity or entities using the PhL transmission procedures. The receipt of this primitive causes the PhL entity to perform all PhL specific actions and pass the PhL service data unit – the received byte – to the physical data interface for transfer to the peer PhL entity or entities.

### 5.2.3.2.2 PH-DATA.indication

#### 5.2.3.2.2.1 Function

This primitive is the service indication primitive of the data transfer service.

#### 5.2.3.2.2.2 Semantics of the service primitive

The semantics of this primitive is as follows:

```
PH-DATA.indication      (  
                           Data  
                         )
```

Where:

- Data carries the received byte as received by the local PhL entity.

#### 5.2.3.2.2.3 Use

The PH-DATA.indication primitive is generated by the PHL entity to indicate to the service user entity the arrival of a valid data byte.

### 5.2.3.3 The PH-ABORT service

#### 5.2.3.3.1 PH-ABORT.request

##### 5.2.3.3.1.1 Function

This primitive is the service request primitive of the connection abort service.

##### 5.2.3.3.1.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
PH-ABORT.request ( )
```

##### 5.2.3.3.1.3 Use

The PH-ABORT.request primitive is invoked by the service user entity Physical Connection Manager to request the PhL entity to terminate an existing physical connection.

#### 5.2.3.3.2 PH-ABORT.confirm

##### 5.2.3.3.2.1 Function

This primitive is the service confirm primitive of the connection abort service.

##### 5.2.3.3.2.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
PH-ABORT.confirm      (  
                           Result  
                         )
```

Where:

- Result carries the result of the physical disconnection attempt.

##### 5.2.3.3.2.3 Use

The PH-ABORT.confirm primitive is generated by the PhL entity to confirm to the service user entity Physical Connection Manager the result of a physical disconnection attempt.

#### 5.2.3.3.3 PH-ABORT.indication

##### 5.2.3.3.3.1 Function

This primitive is the service indication primitive of the connection abort service.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	63/614
-----------------------	------------	-----------------------	--------

### 5.2.3.3.3.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

**PH-ABORT.indication ()**

### 5.2.3.3.3 Use

The PH-ABORT.indication primitive is generated by the PhL entity to inform the service user entity(ies) that a physical connection has been unexpectedly terminated.

## 5.3 Protocol specification

### 5.3.1 Physical layer protocol data unit

The PHPDU is specified to be one byte. For transmission purposes however, this data byte may be extended (error detection / correction) or modified (bit-stuffing) by the modem device, depending on the modulation scheme used. See also explanation to Figure 20.

### 5.3.2 Transmission order and characteristics

The PHSDU byte – the Data parameter of the PH-DATA services – shall be completed with one start bit and one stop bit before transmission. The resulting frame shall be transmitted starting with the start bit, followed by the least significant bit first, with the least significant bit identified as bit 0, and the most significant bit as bit 7.

All characteristics of the physical medium and the signal(s) used on this medium are not in the Scope of this Technical Report.

### 5.3.3 Physical layer operation – description of the procedures

#### 5.3.3.1 General

The PhL – together with the physical media – is a shared resource for the higher protocol layers. Multiple higher layer connections/associations can be modelled as different instances of the protocol stack, which need to share the resources of the PhL.

For this reason, connection establishment and release is managed by the physical connection manager AP – see 5.3.3.2 and 5.3.3.5. Any AP wishing to use the DLMS/COSEM protocol shall check the connection state of the PhL before requesting a connection. If the PhL is in non-connected state, it shall request the physical connection manager to establish the connection. If the AL invokes a COSEM-OPEN.request service and the corresponding physical connection is not established, the COSEM-OPEN.request primitive will be locally confirmed with error = NO\_PHYSICAL\_CONNECTION. See 9.3.2.

Once the physical connection is established, the PhL is ready to transmit data.

An optional Identification service – as described in 5.3.3.3 – is available. This enables the client to identify the protocol stack implemented in the server.

After the identification procedure is completed – or if it is not used – the upper protocol layers and the applications can exchange data – see 5.3.3.4. The user of the PH-DATA services is the next protocol layer above the PhL.

A physical disconnection may be requested by the physical connection manager (either on the server or the client side), or may occur in an unsolicited manner (for example the phone exchange cuts the line).

While physical disconnection management is the exclusive responsibility of the physical connection manager, indication of an unsolicited disconnection (PH-ABORT.indication) is sent both to the next protocol layer and to the physical connection manager. See 5.3.3.5.

#### 5.3.3.2 Setting up a physical connection

Both the client and the server device can act as a calling device, initiating a physical connection to a remote device, which is the called device. In this DLMS/COSEM profile, the service user of these primitives is exclusively the physical connection manager process.

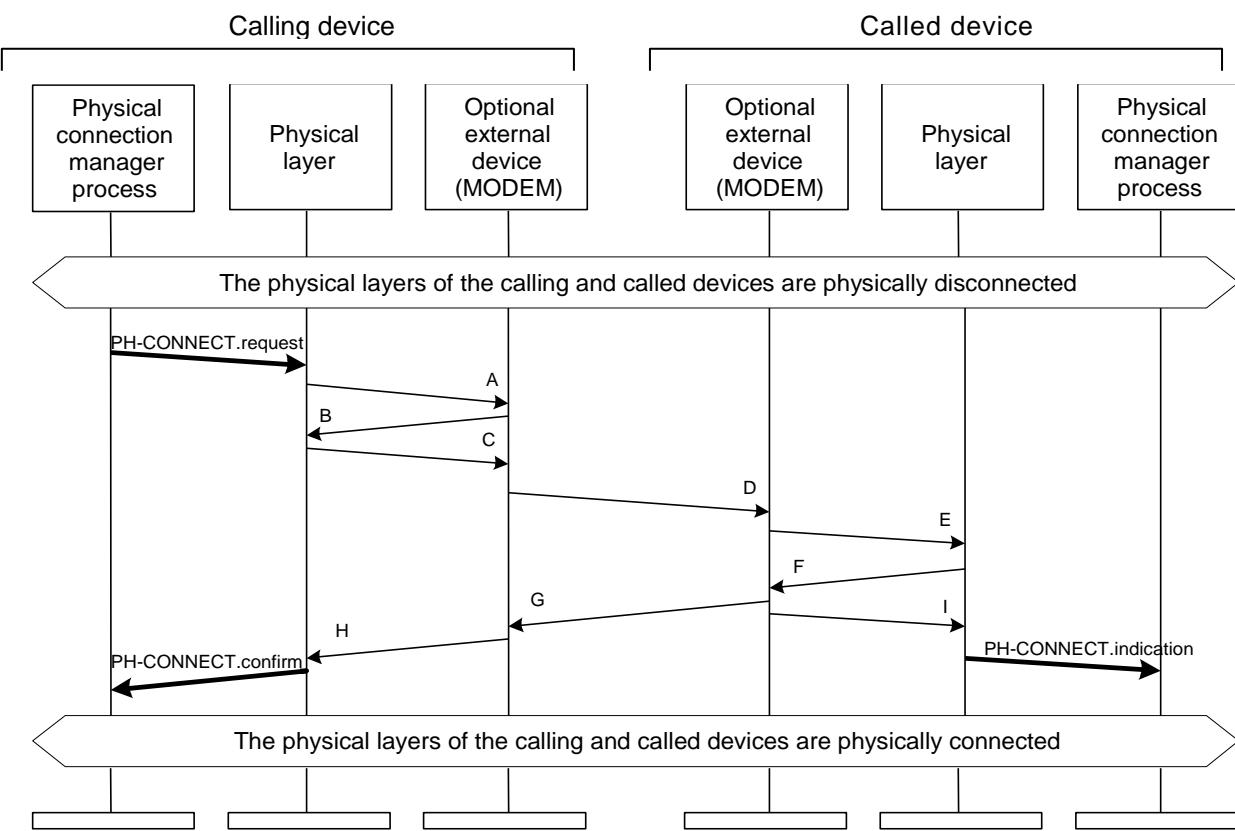
The execution of the PH-CONNECT.request primitive depends on the physical connection type and on the modem used. In 5.4, an example is given in the case when intelligent Hayes modems are used. In other cases, all the operations required – dialling, handling eventual error messages (busy, etc...), negotiating the line modulation / baud-rate parameters, etc. – might be executed by the PhL itself.

In order to allow using a wide variety of physical connection types, this Technical Report does not specify how the execution of the PH-CONNECT.request primitive should be done.

At the called device side, when the physical connection initiation is detected, the connection needs to be managed: negotiated and accepted or refused. These actions – similarly for the execution of the PH-CONNECT.request primitive – depend on the physical connection type and on the modem used, and might be done in an autonomous manner or by the PhL itself. The specification of these actions is not within the Scope of this Technical Report.

When the PhLs of the Calling and Called device complete establishing (or not) the required physical connection, they inform the service user entity about the result, using the PH-CONNECT.confirm (calling side) and the PH-CONNECT.indication (called side) primitives.

As shown in Figure 13, this Technical Report specifies only the PH-CONNECT.request / .confirm / .indication primitives: all other eventual message exchanges (A, B, C,...I) are out of its Scope.



**Figure 13 – MSC for physical connection establishment**

### 5.3.3.3 The Identification service

#### 5.3.3.3.1 General

The optional identification service is an application level service. Its purpose is to allow the client to obtain information about the protocol stack implemented in the server. Consequently, it does not use the whole protocol stack; identification messages are exchanged directly between the client and server APs using the PhL data services. If more than one server is used in a multidrop configuration, the client is able to identify the protocol stack in each.

The identification service shall be the first service after establishing the physical connection. Although the connection may be initiated either by the client or the server, the identification request is always issued by the client.

**NOTE** As the identification service is the first service after establishing a physical connection, the physical connection manager AP could also provide this service.

### 5.3.3.3.2 Identification service specification

#### 5.3.3.3.2.1 IDENTIFY.request

The IDENTIFY.request primitive is invoked by the client AP.

```
IDENTIFY.request ::= (
    IDENTIFY-Request-ID Unsigned8 = 0x202,
    Multidrop-Device-ID OCTET STRING (SIZE (2)) OPTIONAL
)
```

The IDENTIFY-Request-ID parameter identifies the request.

The optional Multidrop-Device-ID parameter addresses one physical device on a multi-drop configuration. Only the addressed device shall respond.

**NOTE** In multidrop configurations, the client has to send the I-command with an address field.

If the server side PhL accepts this message as an identification message, it is indicated to the identification service user AP as an IDENTIFY.indication primitive.

#### 5.3.3.3.2.2 IDENTIFY.response

The IDENTIFY.response message is invoked by the server AP and carries the result of the identification request: the protocol standard, version and revision information or an error message. On the client side, this is an IDENTIFY.confirm primitive.

```
IDENTIFY.response ::= (
    success-code Unsigned8 = <OK> --3)
    std-protocol-id Unsigned8, --3)
    std-protocol-ver Unsigned8, --3)
    std-protocol-rev Unsigned8 --3)
)
```

**NOTE** The response - in case of success - shall be sent with a delay of 1 500 ms maximum.

The following codes shall be used, in conformance with ANSI C12.21:

success-code	- 0x00
std-protocol-id	- 0x04
std-protocol-ver	- 0x01
std-protocol-rev	- 0x00

If there is a problem with the identification message received, the received message shall be discarded and no response shall be sent. Otherwise, the response contains the success code <OK> and the identifier, version and revision of the protocol stack implemented. These identifiers are administrated by the DLMS User Association.

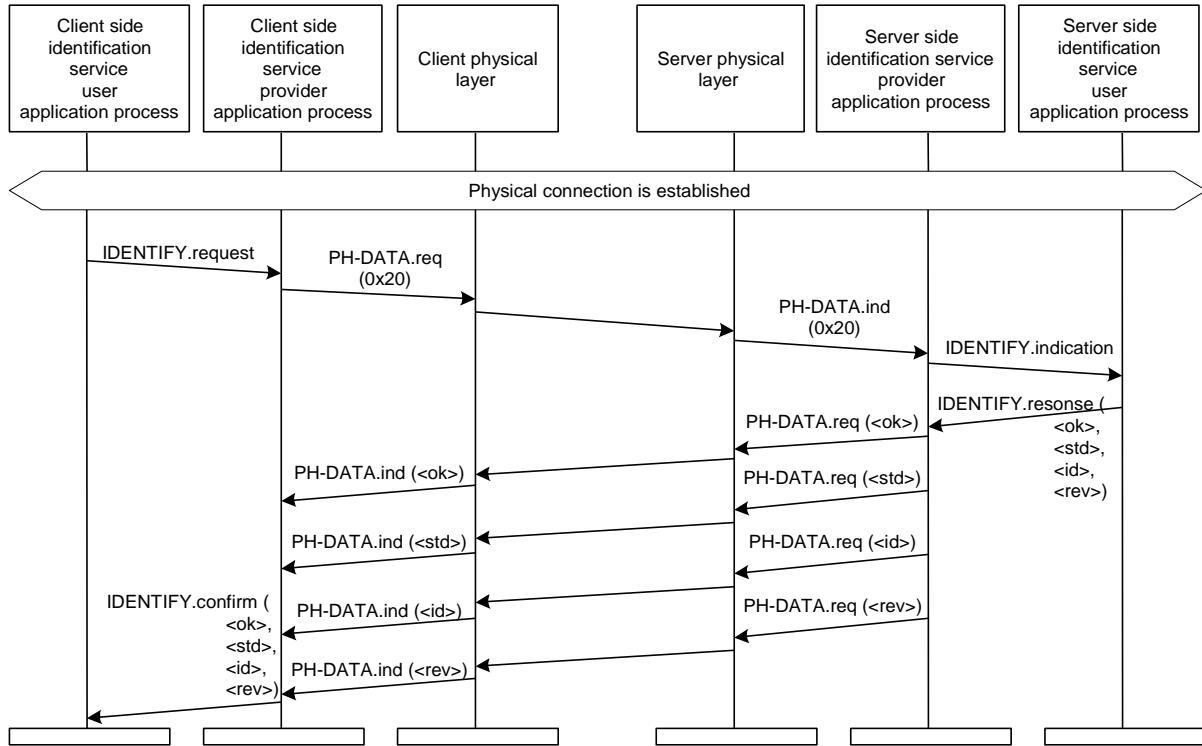
**Important note:** The IDENTIFY.request/.response primitives are not encoded in A-XDR, like other APDUs: they are encoded simply as a sequence of bytes. This means that the IDENTIFY.request / .indication APDU contains one byte or three bytes when the optional multidrop-device-id is present. The IDENTIFY.response / .confirm APDU contains four bytes in case of success.

<sup>2</sup> In order to ensure compliance to existing implementations, the ASCII code of the 'I' character (0x49) may also be used as Identify-Request-ID.

<sup>3</sup> As specified in ANSI C12.21.

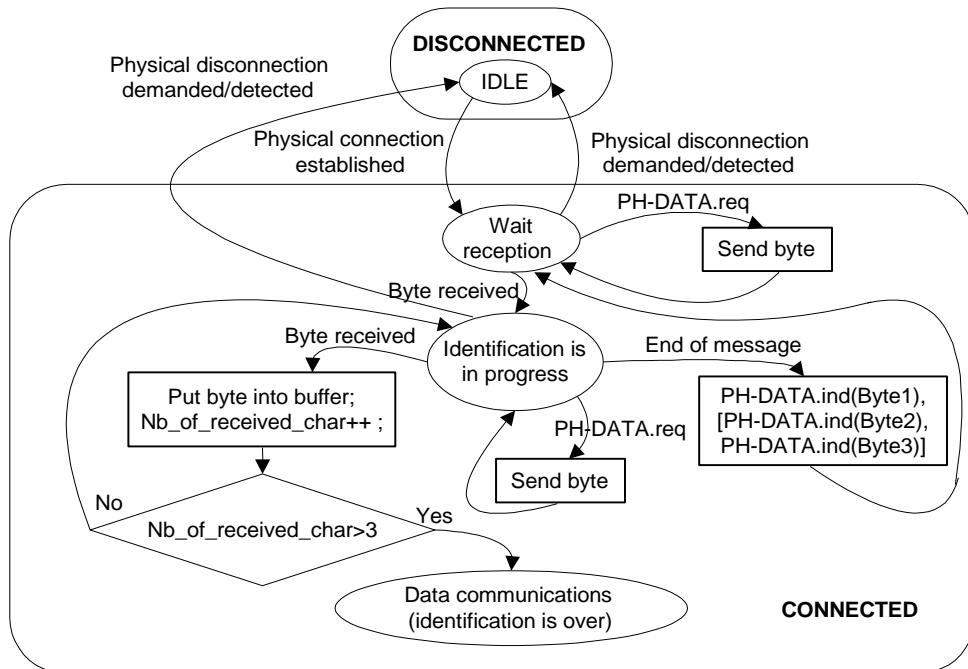
### 5.3.3.3.3 Identification service protocol specification

Figure 14 shows the message sequence chart of the identification service in the case of success.



**Figure 14 – MSC for IDENTIFY.request / .response message exchange**

Figure 15 shows the partial state-machine of the identification service of the server side PhL.



**Figure 15 – Handling the Identification service at the server side**

The server PhL enters the CONNECTED macro-state following the establishment of the physical connection and waits for the first byte of the IDENTIFY.request message in the ‘wait reception’ state.

The IDENTIFY.request APDU contains one or three bytes. For coherency, it shall be sent with the timing constraints of the data link layer (inter-frame- and response time-outs).

When this first character is received, the PhL enters the ‘identification is in progress’ state, waiting for more bytes or an inter-frame time-out, meaning the end of the message.

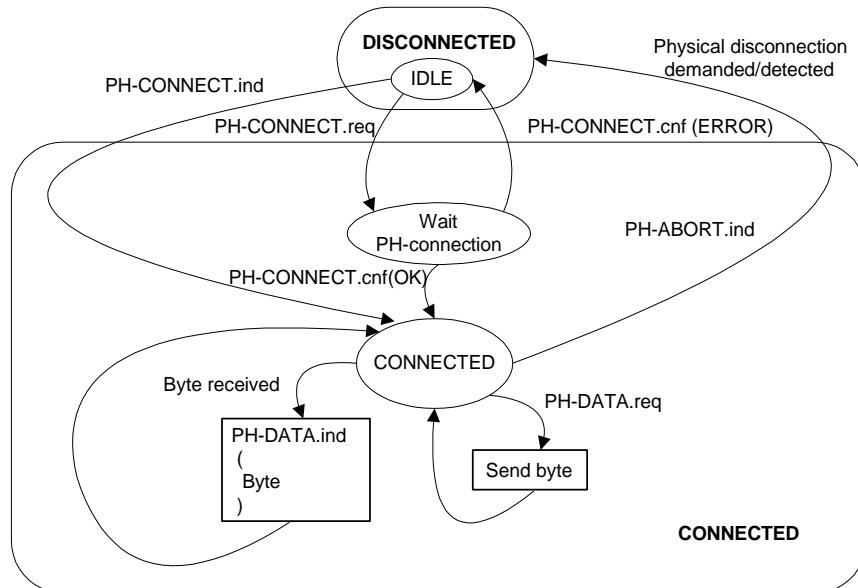
If the end of message condition is detected before receiving more than three bytes, the PhL considers the APDU received as an IDENTIFY.request APDU. It sends the bytes received to the (physical connection manager) AP using the PH-DATA.indication primitive and returns to the ‘wait reception’ state, allowing resolution of eventual errors.

On the other hand, if no end of message condition is detected before receiving the fourth incoming byte, the PhL considers that the identification process is over, and enters into 'data transfer' state. The incoming bytes shall be sent, using the PH-DATA.indication service, to the service user upper protocol layer. In the 3-layer, CO, HDLC based COSEM profile this is the MAC sublayer. Within this connection, the PhL cannot return to the identification stage.

## NOTES:

- 1) The basic assumption of this state machine is that any upper layer PDU (here it is the MPDU) is longer than three characters.
  - 2) The state machine shown in Figure 15 is not complete: for example it is not indicated where the Nb\_of\_received\_char layer parameter is set to its initial value; exit conditions and transitions from the 'data transfer' state are not shown.
  - 3) This identification service definition ensures backward compatibility with client systems, which are not using the optional identification service. If the first message of the client is not an IDENTIFY.request – but longer than three characters – it shall be given to the data link layer and the identification stage is over, too.

The partial state machine for the client side PhL is shown in Figure 16.



**Figure 16 – Partial state machine for the client side physical layer**

The client side PhL uses a layer parameter, 'Destination\_process', to decide where to send the data received. The layer parameter shall be managed by the layer management AP. When this parameter is not set (NULL), the PhL shall send PH-DATA.indication-s to the (physical connection manager) AP. When the identification phase is over, the client AP shall set the 'Destination\_process' parameter to point to the next upper layer of the protocol stack (the MAC sublayer). From this moment PH-DATA.indication primitives (and, in the case of a physical connection interruption, a copy of the PH-ABORT.indication) shall be sent to the upper protocol layer.

### 5.3.3.4 Data transfer

Once the PhL exits from the identification stage, it enters into the data transfer phase, where the PH-DATA.request and PH-DATA.indication primitives are exclusively used by the upper protocol layer, the data link layer.

The PhL is not responsible for any data flow control function: the data received with a PH-DATA.request primitive shall be either transmitted immediately, or – when a physical data flow control is implemented – shall overwrite the previous, not yet transmitted byte. As the PH-DATA service is neither locally nor remotely confirmed, no error shall be signalled in this latter case.

### 5.3.3.5 Disconnection of an existing physical connection

Either the client or the server can initiate the disconnection of an existing physical connection. This takes place by the physical connection manager AP invoking the PH-ABORT.request primitive, as it is shown in Figure 12.

The PhL tries to disconnect the current physical connection and informs the requestor about the result via the PH-ABORT.confirm primitive.

The PH-ABORT.request service is always locally confirmed: the remote unit does not receive any message; it simply detects the disruption of the physical channel (for example the carrier is no longer available).

When the client or the server detects a physical disconnection – this can be the result of a PH-ABORT.request primitive invocation in the remote station but also due to a line error – the PhL shall indicate this event using the PH-ABORT.indication primitive. This shall be sent not only to the physical connection manager AP but also to the next higher protocol layer. This information is necessary for the upper layers to correctly close their connections after the disruption of the physical channel.

## 5.4 Example: PhL service primitives and Hayes commands

### 5.4.1 General

The purpose of this clause is to describe the principles of using an intelligent modem below the PhL interface. It is not the intention to give a complete reference of the Hayes command set or the possibilities provided by the PhL.

The Hayes command set is mainly used in the PSTN modem environment. There is a difference between the command mode and the data transmission mode. In command mode, all commands are issued with a leading "AT". This enables the modem to adapt automatically to the baud rate and line parameters on DTE/DCE side. In data transmission mode, all data is passed to the remote DCE. Data is buffered if an additional data correction or compression mode is enabled.

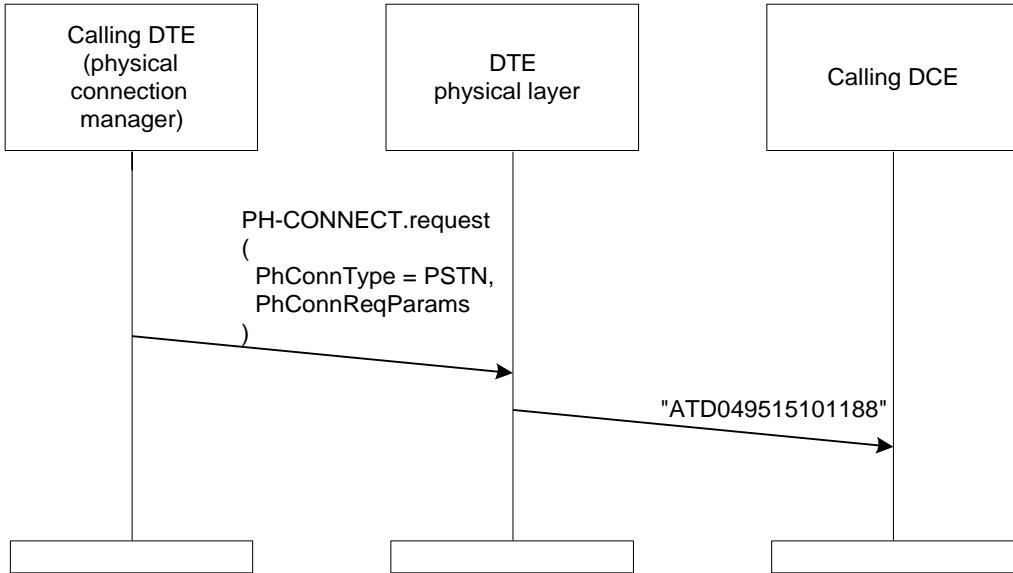
### 5.4.2 Physical layer services and related message exchanges

#### 5.4.2.1 PH-CONNECT.request

The DTE requests to establish a physical connection with a remote DTE by transmitting the dial command together with the phone number to the DCE (ATDPhoneNumber).

Example: ATD049515101188

dials the phone number 049515101188 using the default dialling mode.

**Figure 17 – MSC for physical connection request**

As it is shown, the PhL extracts the telephone number from the PhConnReqParams service parameter, and sends it as a series of ASCII characters headed by the "ATD" Hayes command identifier to the DCE. No more action is required in the PhL – off hook the line, dialling and analysing the result is done by the DCE in an autonomous way (that is why this type of modem is called 'intelligent'). The PhL simply waits the result of the command execution, which is sent back by the DCE in the form of a Hayes message, as it is shown in Figure 18.

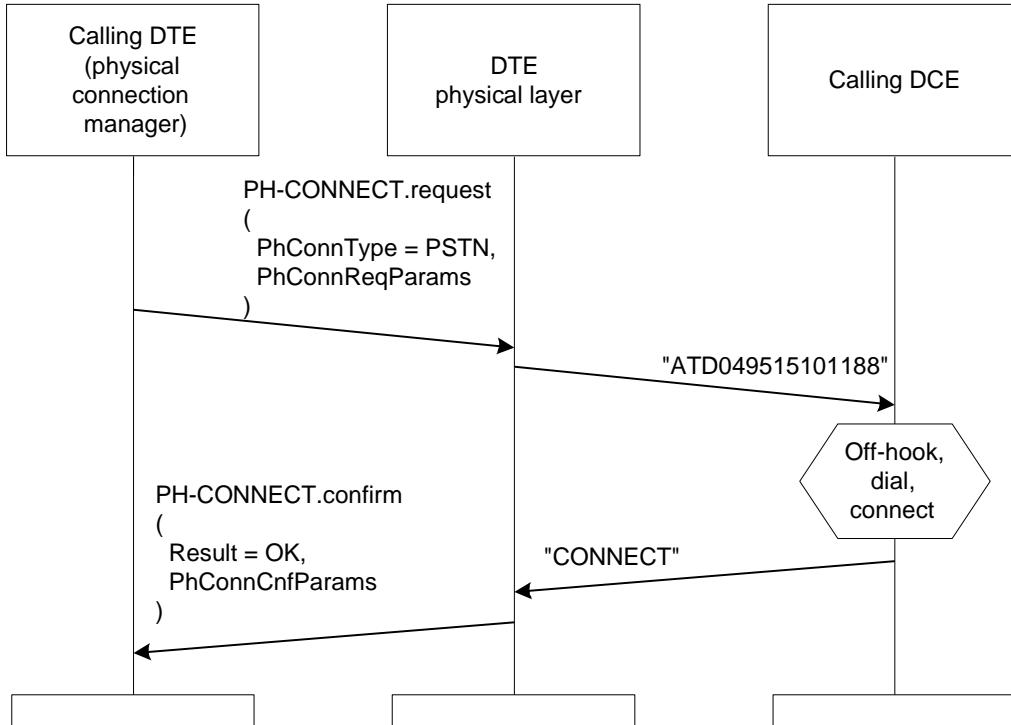
#### 5.4.2.2 PH-CONNECT.confirm

When the DCE is in ASCII mode, it generates one of the following messages after trying to dial the previously received phone number:

- CONNECT: Indicates that the connection has been established successfully. After issuing the connect message, the DCE switches to the data transmission mode;
- ERROR: Indicates a general error or an invalid dial command;
- NO DIALTONE: Indicates that there was no dial tone detected within the given timeout period;
- NO CARRIER: Indicates that the connection has not been established because there was no carrier detected from the remote DCE;
- BUSY: Indicates that the connection has not been established because the remote DCE is busy.

When the PhL receives any of these messages, it generates a PH-CONNECT.confirm primitive with the correct service parameters to the service user, the physical connection manager AP.

Figure 18 shows the complete message sequence at the CALLING station in case of successful physical connection establishment.



**Figure 18 – Physical connection establishment at the CALLING station**

#### 5.4.2.3 PH-CONNECT.indication

A DCE indicates an incoming call by issuing the RING message to the DTE. If the DCE was switched to the AutoAnswer mode during the initialize procedure, it does not send RING messages, but tries to establish the connection automatically after detecting the specified number of ring signals.

If the auto answer mode is disabled, the PHL entity can decide whether to pick up the phone or not by using the "ATA" command. This may be applicable if the server maintains time windows to answer the phone.

Messages to the DTE:

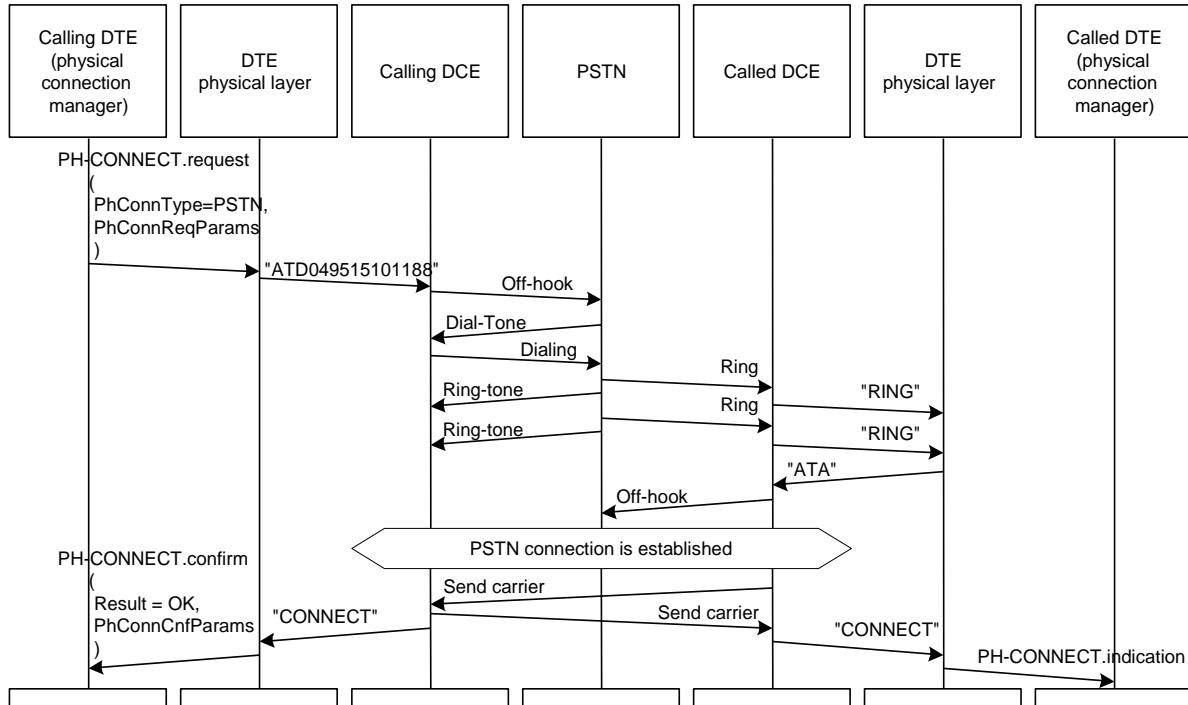
**RING**: Indicates an incoming call.

Commands to the DCE:

**ATA**: Answer the incoming call.

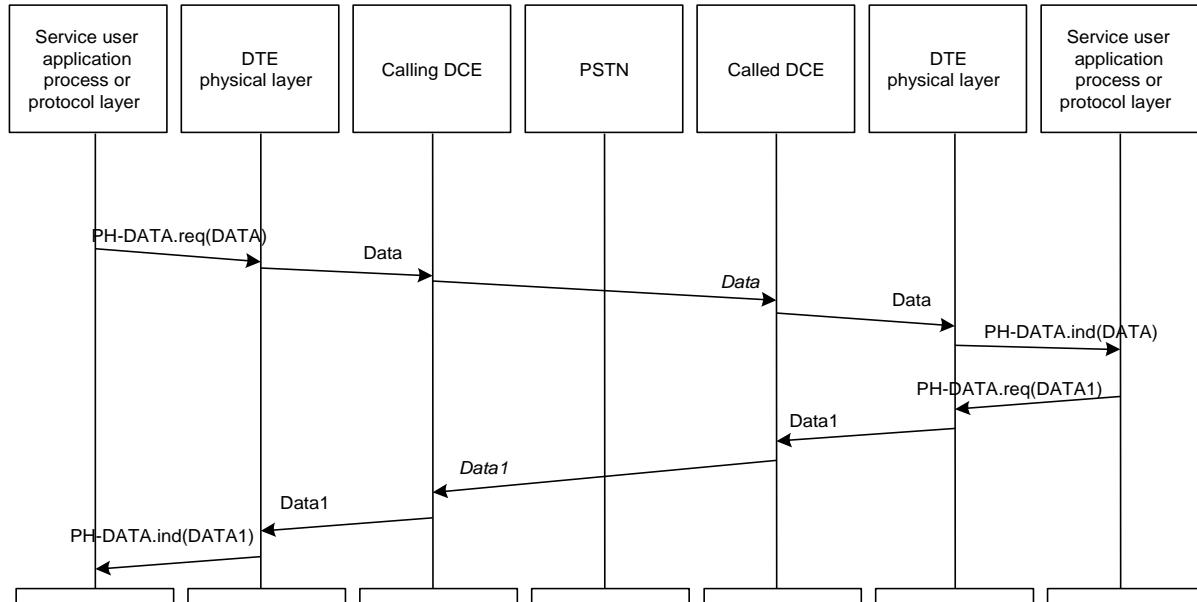
In both cases the DCE signals the result in the same way as it was discussed at PH-CONNECT.confirm.

A simplified message sequence of a complete physical connection establishment is shown in Figure 19, when the DCE is not working in AutoAnswer mode.

**Figure 19 – MSC for physical connection establishment**

#### 5.4.2.4 PH-DATA.request / .indication

Assuming that a connection with a remote DCE was established before and that the DCE is in data transmission mode now, all data passed to the local DCE will be transmitted to the remote DCE.

**Figure 20 – Data exchange between the calling and called stations**

Please note that there is a difference between the actual data formats:

- "DATA/(DATA1)", the service parameter of the PH-DATA service is specified in 5.3.1 as one byte;
- "Data/(Data1)" is a frame containing the data with some additional bits for the asynchronous transmission (start and stop bit);
- "Data/(Data1)", the information exchanged between the two DCEs might contain even more bits for the purposes of error detection/correction functions) and/or could be encoded for the data transmission purposes.

#### 5.4.2.5 PH-ABORT.request / .confirm

Before the connection can be terminated, the modem has to be switched to local command mode first. For this purpose the Hayes environment provides an Escape sequence defined as:

- 1 second idle state (no data transmission);
- sending 3 plus "+++" characters;
- 1 second idle state again.

**NOTE** Most modems allow specifying the value of the Escape Sequence Character (S2 register). The usual value is "+", 43D.

The DCE confirms the command mode with an OK message (but the connection is still valid). The connection can be terminated now using the OnHook "ATH" command.

**Messages to the DTE: OK:** DCE is in command mode again.

**Commands to the DCE: ATH:** terminate the connection (OnHook)

If the OnHook command was successful, the DCE responds with the NO\_CARRIER message, otherwise an ERROR message is returned:

- NO\_CARRIER: connection was successfully aborted;
- ERROR: the OnHook command failed, connection is still available.

#### 5.4.2.6 PH-ABORT.indication

If the carrier is lost, the local DCE issues the "NO\_CARRIER" message. The DTE is not able to determine the reasons for losing the carrier; one of the reasons can be that the remote DTE terminated the connection. Once the PhL has received this message, it should use the PH-ABORT.indication service primitive to indicate the termination of the connection.

**NO\_CARRIER:** connection was aborted by the remote DCE or due an error condition.

Figure 21 shows an example for physical disconnection.

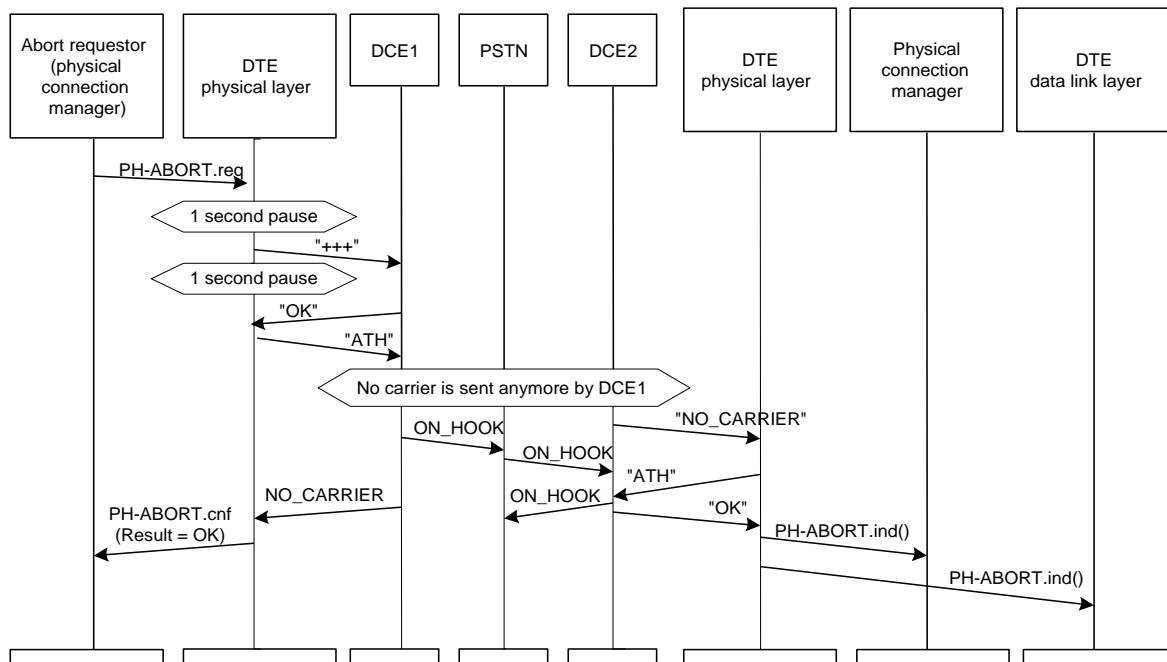


Figure 21 – MSC for a physical disconnection

## 6 Direct Local Connection

### 6.1 Introduction

This Clause 6 is an excerpt of IEC 62056-21 describing hardware and protocol specifications for local meter data exchange. In such systems, a hand-held unit (HHU) or a unit with equivalent functions is connected to a tariff device or a group of devices. Only COSEM related items are described here. The complete information can be found in IEC 62056-21.

**NOTE** Support for local interface based on IEC 62056-21 is not mandated within DLMS/COSEM. Local connection using HDLC *ab initio*, or PPP, or no local interface, are equally acceptable.

### 6.2 METERING HDLC protocol using protocol mode E for direct local data exchange

The protocol stack as described in Clauses 5, 8 and 9 of this Technical Report shall be used.

The switch to the baud rate Z shall be at the same place as for protocol mode C. The switch confirm message, which has the same structure as the acknowledgement-option select message, is therefore at the new baud rate but still with parity (7E1). After the acknowledgement, the binary mode (8N1) will be established.

As the server acknowledgement string is a constant in the server's program, it could be easily possible to switch to the baud rate and the binary mode (Z Bd. 8N1) at the same time. The characters ACK 2 Z 2 CR LF in that case shall be replaced by their 8 bit equivalents by adding the correct parity bit in order to simulate their 7E1 equivalents. This alternative method is not visible to the client; both have an equivalent behaviour (see also Figure 25).

A client, which is not able to support protocol HDLC mode E (W=2) will answer in a protocol mode as defined by Y (normally protocol mode C).

The enhanced capability of the server (tariff device) is communicated with the escape sequence "\W" which is part of the meter identification string (see items 14), 23) and 24) in IEC 62056-21:2002, 6.3.14<sup>4</sup>.

### 6.3 Overview

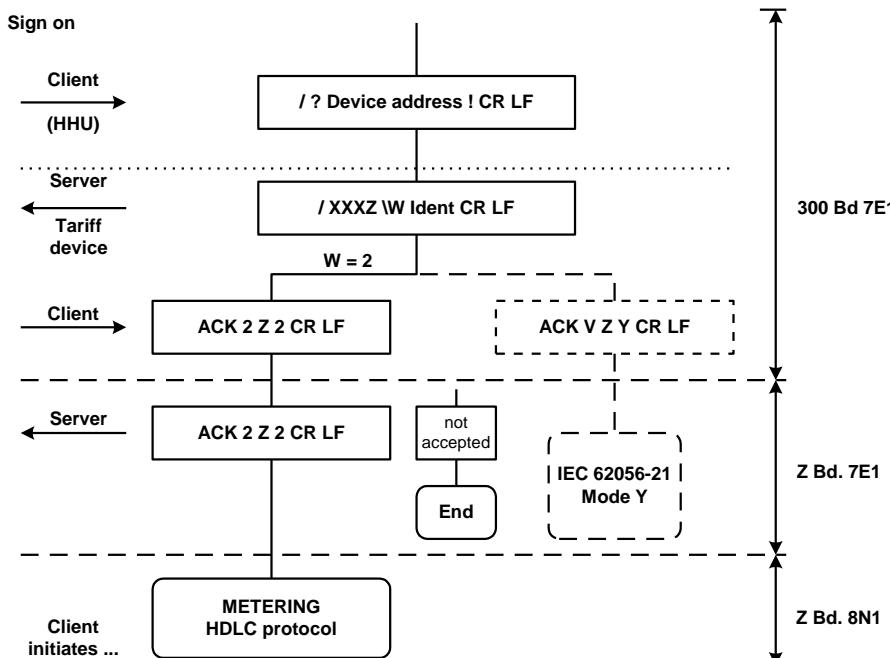


Figure 22 – Entering protocol mode E (HDLC)

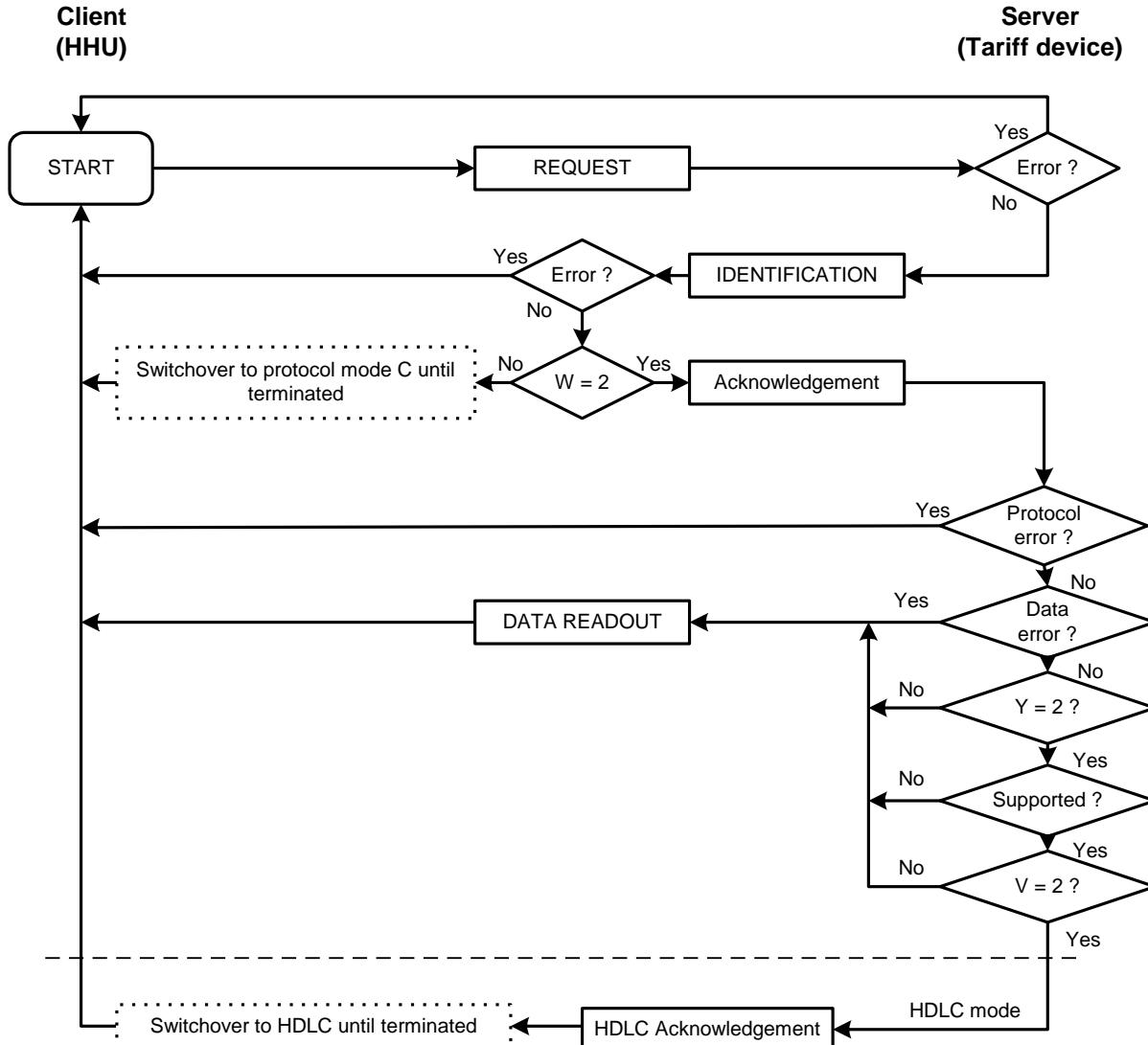
<sup>4</sup> W = @ is used for country specific applications

## 6.4 Readout mode and programming mode

These modes are handled within the higher layers of the protocol. After having established a transparent channel, the "METERING HDLC protocol" takes care of the correct data handling, and a DLMS/COSEM based application handles access rights, read only or read/write access etc. Necessary procedures are described in chapters 5, 8 and 9.

The flow chart and the changeover to HDLC for the direct local data exchange protocol, protocol mode E is shown below.

**NOTE** On disconnection of the data link a server implementing the IEC 62056-21 protocol will revert to the initial state without requiring the signoff message.



**Figure 23 – Flow chart and switchover to METERING HDLC in protocol mode E**

### Key to protocol mode E flow diagram

Message formats

REQUEST

/? Device Address ! CR LF

IDENTIFICATION

/ XXX Z Ident CR LF

Acknowledgement

ACK 2 Z 2 CR LF

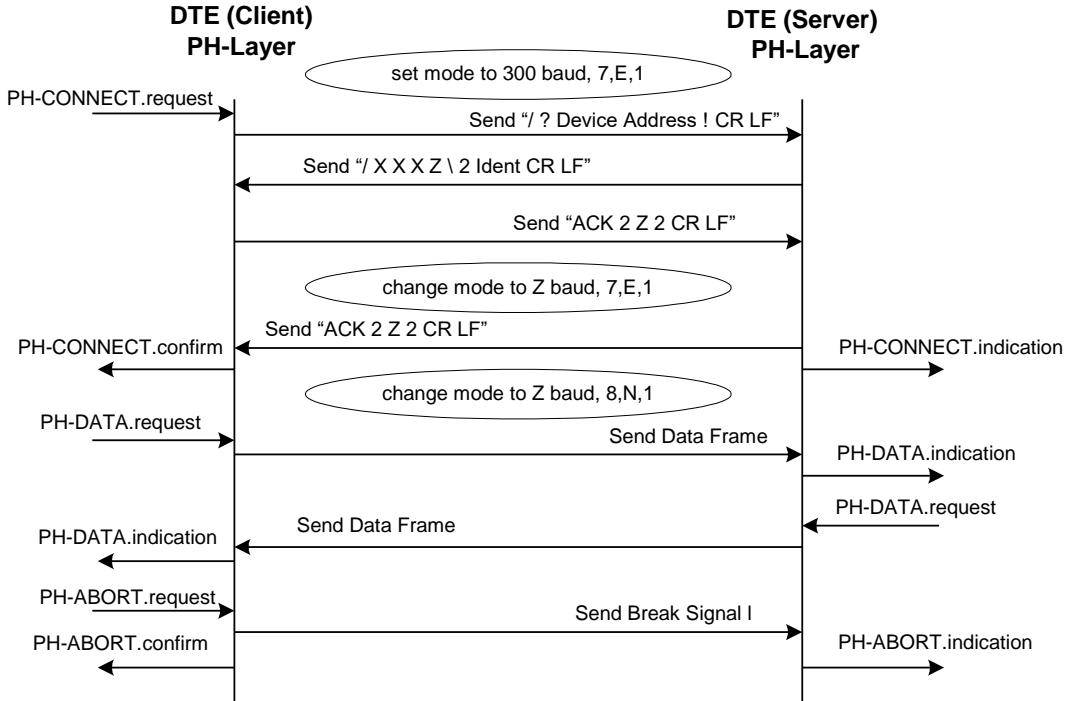
DATA READOUT (fall back data readout mode A)

STX DATA ! CR LF ETX BCC

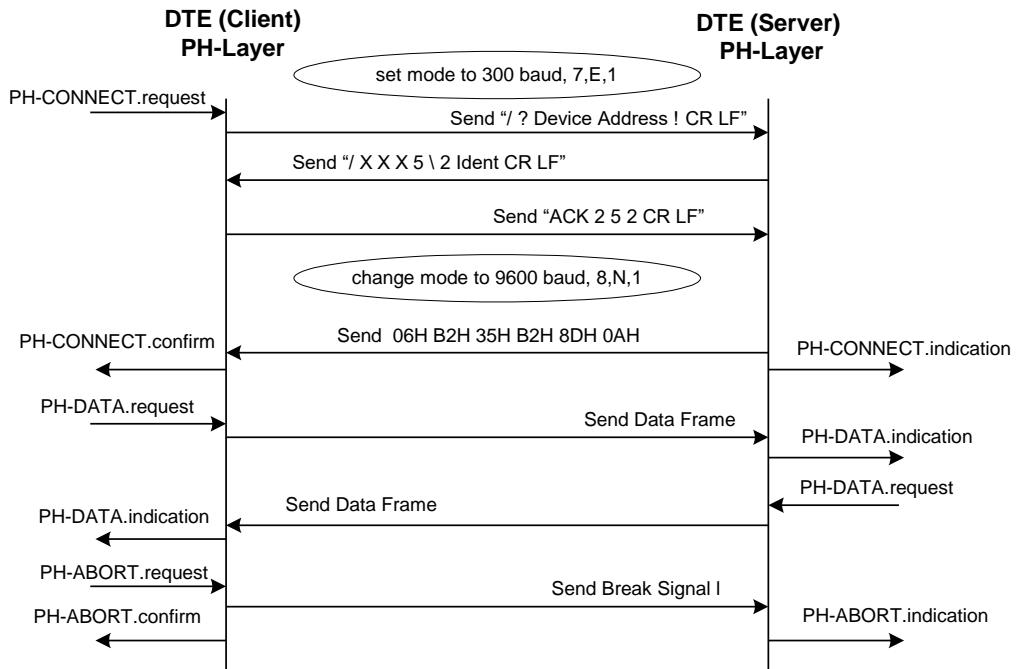
**NOTE** The inactivity time-out period for the tariff device is 60 s to 120 s after which the operation moves from any point to the start.

## 6.5 Physical layer – Introduction

The framework is equivalent to "Physical layer services and procedures for connection-oriented asynchronous data exchange" (see Clause 5).



**Figure 24 – Physical layer primitives**



**Figure 25 – Physical layer primitives, simplified example with one mode change only**

## 6.6 Physical layer primitives

### PH-CONNECT.request

Once the PH-CONNECT.request primitive has been invoked with this connection type, the PhL entity will start to establish the connection according to the procedure described above. The device address is passed via the PhConnType parameter. For this purpose a mapping of the Lower MAC address to the device address (IEC 62056-21, item 22, 6.3.14) has to be specified. Note, that a PH-CONNECT.request primitive cannot be invoked by the server (tariff device).

### PH-CONNECT.confirm

After receiving the ACK 2 Z 2 CR LF or other, for example NAK message from the server (tariff device), the PH-CONNECT.confirm primitive is generated with the appropriate result parameter.

Messages:

ACK 2 Z 2 CR LF the metering device has entered the METERING HDLC protocol mode E

Other response the PH-CONNECT.request failed

### PH-CONNECT.indication

After the server's PH-Layer has acknowledged the METERING HDLC protocol mode E, it indicates this to the MAC-sublayer by generating the PH-CONNECT.indication primitive. During HDLC operation, timeouts etc. are following HDLC rules.

### PH-ABORT.request

The PhL entity aborts the connection.

NOTE BREAK is only local to the client, the server does not respond, timeout is used. Timeouts for HDLC are defined in Clause 8.

### PH-ABORT.confirm

Since the client will never receive a response from the server, the PhL entity always has to confirm the PH-ABORT.request primitive.

NOTE BREAK is only local to the client, the server does not respond, timeout is used. Timeouts for HDLC are defined in Clause 8.

### PH-ABORT.indication

Detecting BREAK, the server PhL entity resets its state machine to the initial state and invokes the PH-ABORT.indication service to indicate the termination of the connection.

## 6.7 Data link layer

See Clause 8.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	77/614
-----------------------	------------	-----------------------	--------

## 7 DLMS/COSEM transport layer for IP networks

### 7.1 Scope

This clause specifies

- a connection-less UDP based transport layer;
- a connection oriented TCP based transport layer;
- a connection-less CoAP based transport layer

for DLMS/COSEM communication profiles used on IP networks.

The TCP-UDP/IP based transport layers are specified in 7.2. These transport layers are provided for the DLMS/COSEM\_on\_IP communication profiles.

The CoAP based transport layer is specified in 7.3. This transport layer is provided for the DLMS/COSEM\_on\_CoAP communication profiles.

The DLMS/COSEM TL consists of the CoAP, UDP or TCP transport layer and an additional sublayer, called wrapper.

Subclause 7.2.5 shows how the OSI-style TL services can be converted to and from UDP and TCP function calls.

### 7.2 The TCP-UDP/IP based transport layers

#### 7.2.1 Scope

This subclause 7.2 specifies a connection-less and a connection oriented transport layer (TL) for the DLMS/COSEM\_on\_IP communication profiles used on IP networks.

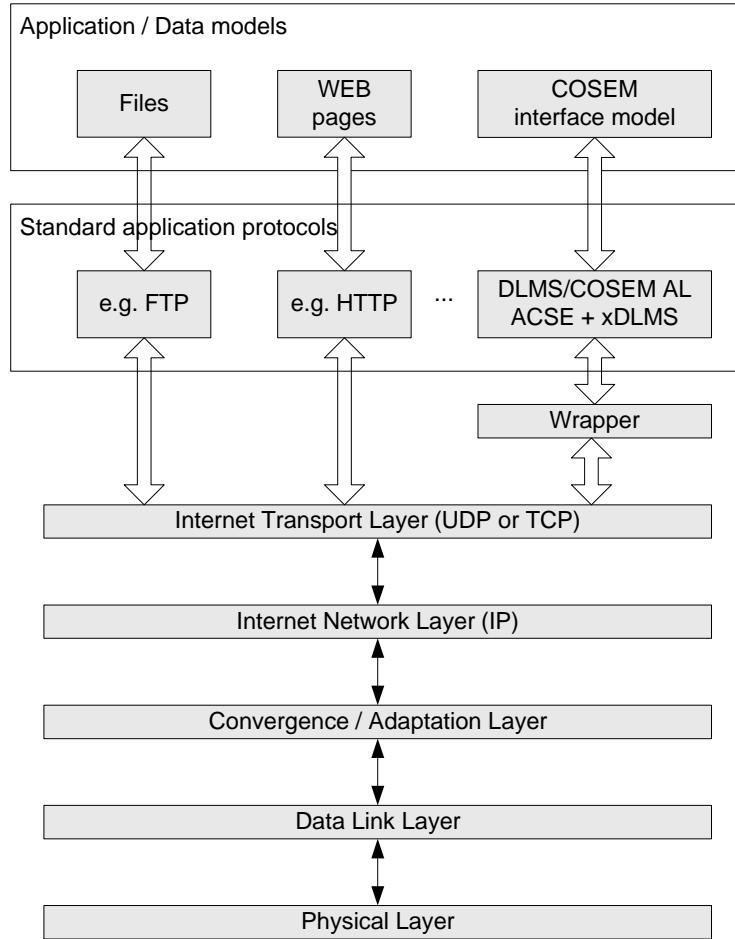
These TLs provide OSI-style services to the service user DLMS/COSEM AL. The connection-less TL is based on the Internet Standard User Datagram Protocol (UDP). The connection-oriented TL is based on the Internet Standard Transmission Control Protocol (TCP).

Subclause 7.2.5 describes how the OSI-style TL services can be converted to and from UDP and TCP function calls.

#### 7.2.2 Overview

In the DLMS/COSEM\_on\_IP profiles, the DLMS/COSEM AL uses the services of one of these TLs, which use then the services of the Internet Protocol (IP) network layer to communicate with other nodes connected to the IP network.

When used in these profiles, the DLMS/COSEM AL can be considered as another Internet standard application protocol (like the well-known HTTP, FTP or SNMP) and it may co-exist with other Internet application protocols, as it is shown in Figure 26.



**Figure 26 – DLMS/COSEM as a standard Internet application protocol**

For DLMS/COSEM, the following port numbers have been registered by the IANA. See <http://www.iana.org/assignments/port-numbers>:

- dlms/cosem                  4059/TCP        DLMS/COSEM;
- dlms/cosem                  4059/UDP        DLMS/COSEM.

As the DLMS/COSEM AL specified in Clause 9 uses and provides OSI-style services, a wrapper has been introduced between the UDP/TCP layers and the DLMS/COSEM AL. Therefore, the DLMS/COSEM TLs consist of a wrapper sublayer and the UDP or TCP TL. The wrapper sublayer is a lightweight, nearly state-less entity: its main function is to adapt the OSI-style service set, provided by the DLMS/COSEM TL, to UDP or TCP function calls and vice versa.

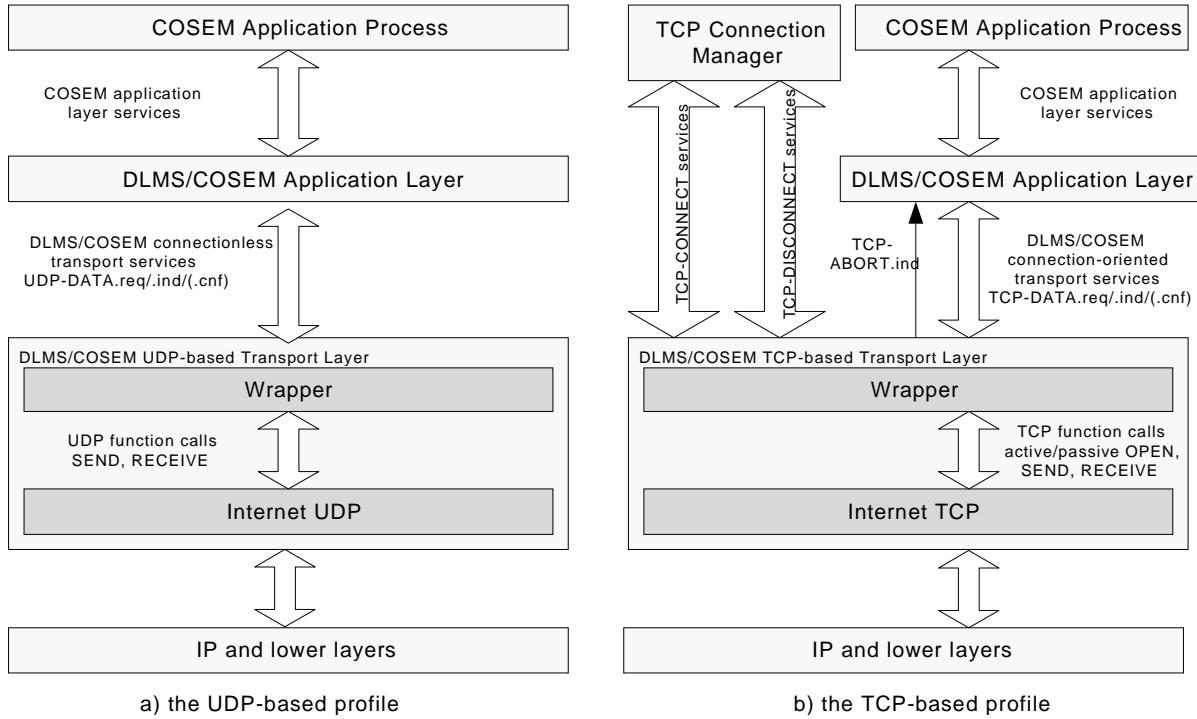
In addition, the wrapper sublayer has the following functions:

- it provides an additional addressing capability (wPort) on top of the UDP/TCP port;
- it provides information about the length of the data transported. This feature helps the sender to send and the receiver to recognize the reception of a complete APDU, which may be sent and received in multiple TCP packets.

As specified in 10.3.3, the DLMS/COSEM AL is listening only on one UDP or TCP port. On the other hand, as shown in 4.9 and in DLMS UA 1000-1, a physical device may host several client or server APs. The additional addressing capability provided by the wrapper sublayer allows addressing these APs.

The structure of the DLMS/COSEM TL and their place in DLMS/COSEM\_on\_IP is shown in Figure 27.

**Figure 27 – Transport layers of the DLMS/COSEM\_on\_IP profile**



The service user of both the UDP-DATA and the TCP-DATA services is the DLMS/COSEM AL. On the other hand, the service user of the TCP-CONNECT and TCP-DISCONNECT services is the TCP Connection Manager Process. The DLMS/COSEM TCP-based TL also provides a TCP-ABORT service to the service user DLMS/COSEM AL.

### 7.2.3 The DLMS/COSEM connection-less, UDP-based transport layer

#### 7.2.3.1 General

The DLMS/COSEM connection-less TL is based on the User Datagram Protocol (UDP) as specified in STD0006.

UDP provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. On the one hand, the protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. On the other hand, UDP is simple, it adds a minimum of overhead, and it is efficient and easy to use. Several well-known Internet applications, like SNMP, DHCP, TFTP, etc. take advantage of these performance benefits, either because some datagram applications do not need to be reliable or because the required reliability mechanism is ensured by the application itself. Request/response type applications, like a confirmed COSEM application association established on the DLMS/COSEM UDP-based TL, then invoking confirmed xDLMS data transfer services is a good example for this second category. Another advantage of UDP is that being connection-less, it is easily capable of multi- and broadcasting.

UDP basically provides an upper interface to the IP layer, with an additional identification capability, the UDP port number. This allows distinguishing between APs, hosted in the same physical device and identified by its IP address.

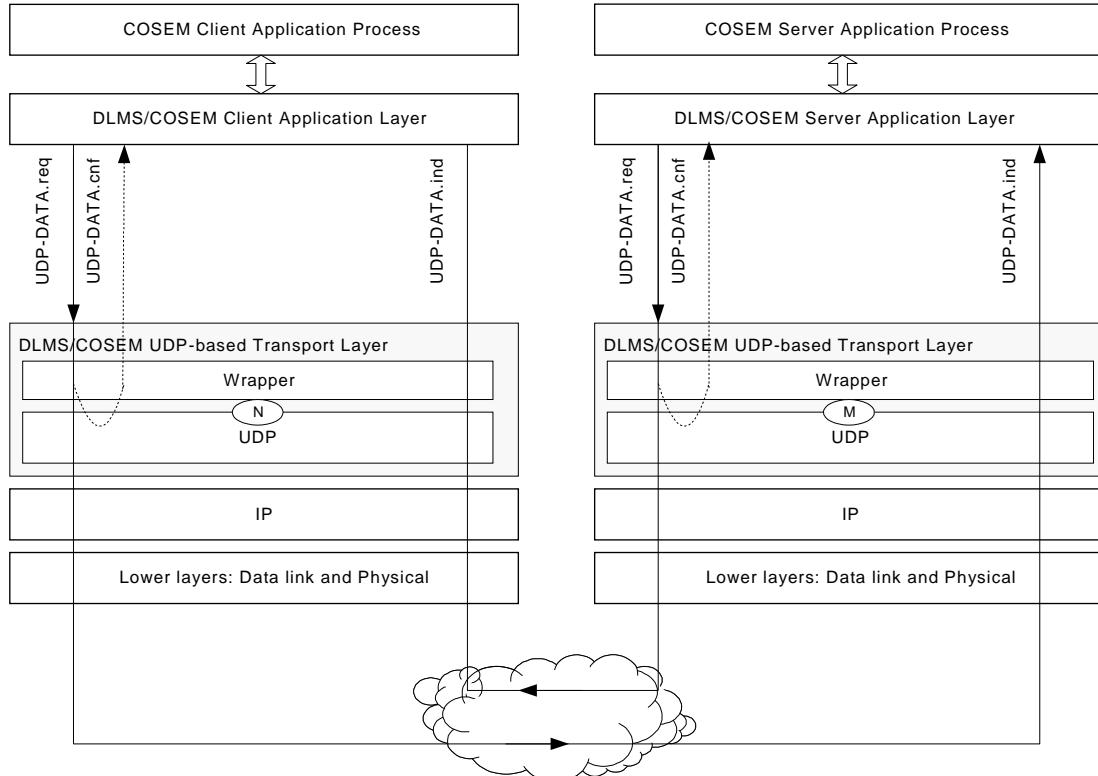
NOTE The addressing/identification scheme for the COSEM\_on\_IP profiles is defined in 10.3.3.

#### 7.2.3.2 Service specification for the DLMS/COSEM UDP-based transport layer

##### 7.2.3.2.1 General

The DLMS/COSEM UDP-based TL provides only a data transfer service: the connection-less UDP-DATA service. Consequently, the service specification for this service is the same for both the client and server TLs, as it is shown in Figure 28.

80/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
--------	------------	-----------------------	-----------------------



**Figure 28 – Services of the DLMS/COSEM connection-less, UDP-based transport layer**

The .request and .indication service primitives are mandatory. The implementation of the local .confirm service primitive is optional.

The xDLMS APDU pre-fixed with the wrapper header shall fit in a single UDP datagram.

#### 7.2.3.2.2 The UDP-DATA service

##### 7.2.3.2.2.1 UDP-DATA.request

###### 7.2.3.2.2.1.1 Function

This primitive is the service request primitive for the connection-less mode data transfer service.

###### 7.2.3.2.2.1.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
UDP-DATA.request ( 
    Local_wPort,
    Remote_wPort,
    Local_UDP_Port,
    Remote_UDP_Port,
    Local_IP_Address,
    Remote_IP_Address,
    Data_Length,
    Data
)
```

Where:

- Local\_wPort, Local\_UDP\_Port and Local\_IP\_Address indicate wrapper Port number, UDP Port number and IP address parameters belonging to the device / DLMS/COSEM AE requesting to send the Data. The Remote\_wPort, Remote\_UDP\_Port and Remote\_IP\_Address parameters indicate the wrapper Port number, UDP Port number and IP address parameters belonging to the device / DLMS/COSEM AE to which the Data is to be transmitted.

- Data\_Length indicates the length of the Data parameter in bytes.
- Data contains the xDLMS APDU to be transferred to the peer AL.

#### 7.2.3.2.2.1.3 Use

The UDP-DATA.request primitive is invoked by either the client or the server DLMS/COSEM AL to request sending an APDU to a single peer AL, or, in the case of multi- or broadcasting, to multiple peer ALs.

The reception of this service primitive shall cause the wrapper sublayer to pre-fix the wrapper header to the APDU received, and then to call the SEND() function of the UDP sublayer with the properly formed WPDU, see at 7.2.3.3.2, as DATA. The UDP sublayer shall transmit the WPDU to the peer wrapper sublayer as described in STD0006.

#### 7.2.3.2.2.2 UDP-DATA.indication

##### 7.2.3.2.2.2.1 Function

This primitive is the service indication primitive for the connection-less mode data transfer service.

##### 7.2.3.2.2.2.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
UDP-DATA.indication (
    Local_wPort,
    Remote_wPort,
    Local_UDP_Port,
    Remote_UDP_Port,
    Local_IP_Address,
    Remote_IP_Address,
    Data_Length,
    Data
)
```

Where:

- Local\_wPort, Local\_UDP\_Port and Local\_IP\_Address indicate wrapper Port number, UDP Port number and IP address parameters belonging to the device / DLMS/COSEM AE receiving the Data. The Remote\_wPort, Remote\_UDP\_Port and Remote\_IP\_Address parameters indicate the wrapper Port number, UDP Port number and IP address parameters belonging to the device / DLMS/COSEM AE, which has sent the data.
- Data\_Length indicates the length of the Data parameter in bytes.
- Data contains the xDLMS APDU received from the peer AL.

#### 7.2.3.2.2.2.3 Use

The UDP-DATA.indication primitive is generated by the DLMS/COSEM UDP based TL to indicate to the service user DLMS/COSEM AL that an APDU from the peer layer entity has been received.

The primitive is generated following the reception of an UDP Datagram by the UDP sublayer, if both the Local\_UDP\_Port and Local\_wPort parameters of the message received contain valid port numbers, meaning that there is a DLMS/COSEM AE in the receiving device bound to the given port numbers. Otherwise, the message received shall simply be discarded.

#### 7.2.3.2.2.3 UDP-DATA.confirm

##### 7.2.3.2.2.3.1 Function

This primitive is the optional service confirm primitive for connection-less mode data transfer service.

##### 7.2.3.2.2.3.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
UDP-DATA.confirm (
    Local_wPort,
```

82/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
--------	------------	-----------------------	-----------------------

```
    Remote_wPort,  
    Local_UDP_Port,  
    Remote_UDP_Port,  
    Local_IP_Address,  
    Remote_IP_Address,  
    Result  
)
```

Where:

Local\_wPort, Remote\_wPort, Local\_UDP\_Port, Remote\_UDP\_Port, Local\_IP\_Address and Remote\_IP\_Address carry the same values as the corresponding UDP-DATA.request service being confirmed.

The value of the Result parameter indicates whether the DLMS/COSEM UDP-based TL was able to send the requested UDP Datagram (OK) or not (NOK).

#### 7.2.3.2.2.3.3 Use

The UDP-DATA.confirm primitive is optional. If implemented, it is generated by the DLMS/COSEM TL to confirm to the service user DLMS/COSEM AL the result of the previous UDP-DATA.request. It is locally generated and indicates only whether the Data in the .request primitive could be sent or not. In other words, an UDP-DATA.confirm with Result == OK means only that the Data has been sent, and does not mean that the Data has been (or will be) successfully delivered to the destination.

### 7.2.3.3 Protocol specification for the DLMS/COSEM UDP-based transport layer

#### 7.2.3.3.1 General

As it is shown in Figure 27, the DLMS/COSEM UDP-based TL includes the Internet Standard UDP layer, as specified in Internet Standard STD0006, and the DLMS/COSEM-specific light-weight wrapper sublayer.

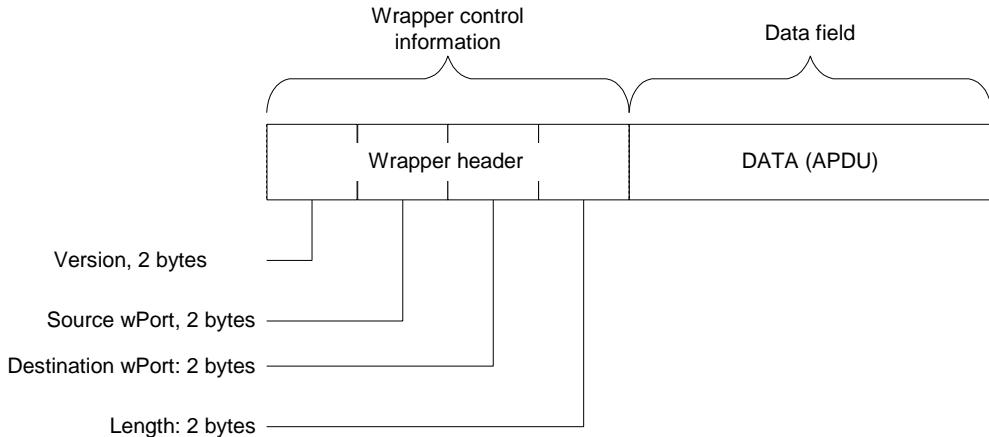
In this communication profile, the wrapper sublayer is a state-less entity: its only roles are to ensure source and destination DLMS/COSEM AE identification using the wPort numbers and to provide conversion between the OSI-style UDP-DATA.xxx service invocations and the SEND() and RECEIVE() interface functions provided by the standard UDP.

Although it is not necessary in the UDP-based profile, in order to have the same wrapper protocol control information – in other words the wrapper header – in both TLs, the wrapper sublayer shall also include the Data Length information in the wrapper protocol data unit.

#### 7.2.3.3.2 The wrapper protocol data unit (WPDU)

The WPDU consists of two parts:

- the wrapper header part, containing the wrapper control information; and
- the data part, containing the DATA parameter – an xDLMS APDU – of the corresponding UDP-DATA.xxx service invocation.



NOTE The maximum length of the APDU should be eight bytes less than the maximum length of the UDP datagram.

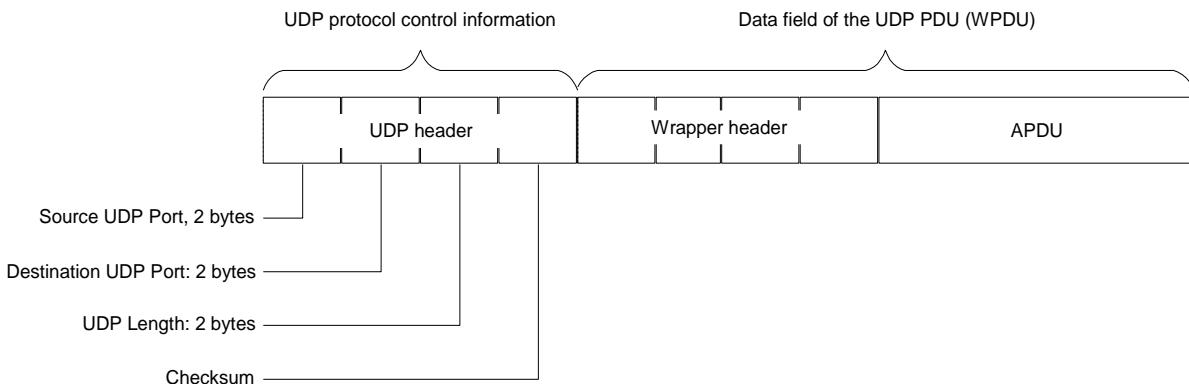
**Figure 29 – The wrapper protocol data unit (WPDU)**

The wrapper header includes four fields, see Figure 29. Each field is a 16 bit long unsigned integer value.

- Version: carries the version of the wrapper. Its value is controlled by the DLMS UA. The current value is 0x0001. Note, that in later versions the wrapper header may have a different structure;
- Source wPort: carries the wPort number identifying the sending DLMS/COSEM AE;
- Destination wPort: carries the wPort number identifying the receiving DLMS/COSEM AE;
- Data length: indicates the length of the DATA field of the WPDU (the xDLMS APDU transported).

#### 7.2.3.3.3 The DLMS/COSEM UDP-based transport layer protocol data unit

In this profile, WPDUs shall be transmitted in UDP Datagrams, specified in Internet standard STD0006. They shall encapsulate the WPDU, as shown in Figure 30.



**Figure 30 – The DLMS/COSEM connection-less, UDP-based transport layer PDU (UDP-PDU)**

From the external point of view, the DLMS/COSEM connection-less TL PDU is an ordinary UDP Datagram: any DLMS/COSEM specific element, including the wrapper-specific header is inside the UDP Data field. Consequently, standard UDP implementations can be (re-)used to easily implement this TL.

The Source and Destination UDP ports may refer to either local or remote UDP ports depending on the direction of the data transfer: from the point of view of the sending device the Source UDP port in a Datagram corresponds to the Local\_UDP\_port, but from the point of view of the receiving device the Source UDP port in a Datagram corresponds to the Remote\_UDP\_Port service parameter.

According to the UDP specification, filling the source UDP Port and Checksum fields with real data is optional. A zero value – all bits are equal to zero – of these fields indicates that in the given UDP

Datagram the field is not used. However, in the DLMS/COSEM\_on\_IP profile, the source UDP Port field shall always be filled with the source UDP port number.

#### 7.2.3.3.4 Reserved wrapper port numbers (wPort)

Reserved wPort Numbers are specified in Table 2:

**Table 2 – Reserved wrapper port numbers in the UDP-based DLMS/COSEM TL**

<b>Client side reserved addresses</b>	
	Wrapper Port Number
No-station	0x0000
Client Management Process	0x0001
Public Client	0x0010
<i>Open for client SAP assignment</i>	0x02 ... 0x0F
	0x11... 0xFF
<b>Server side reserved addresses</b>	
	Wrapper Port Number
No-station	0x0000
Management Logical Device	0x0001
Reserved	0x0002...0x000F
<i>Open for server SAP assignment</i>	0x0010...0x007E
All-station (Broadcast)	0x007F

#### 7.2.3.3.5 Protocol state machine

As the wrapper sublayer in this profile is state-less, for all other protocol related issues – protocol state machine, etc. – the governing rules are as they are specified in the Internet standard STD0006. The only supplementary rule is concerning discarding inappropriate messages: messages with an invalid destination wPort number – meaning that there is no DLMS/COSEM AE in the receiving device bound to this wPort number – shall be discarded by the wrapper sublayer.

### 7.2.4 The DLMS/COSEM connection-oriented, TCP-based transport layer

#### 7.2.4.1 General

The DLMS/COSEM connection-oriented TL is based on the connection-oriented Internet transport protocol, called Transmission Control Protocol. TCP is an end-to-end reliable protocol. This reliability is ensured by a conceptual “virtual circuit”, using a method called PAR, Positive Acknowledgement with Retransmission. It provides acknowledged data delivery, error detection and data re-transmission after an acknowledgement time-out, etc. Therefore it deals with lost, delayed, duplicated or erroneous data packets. In addition, TCP offers an efficient flow control mechanism and full-duplex operation, too.

TCP, as a connection-oriented transfer protocol involves three phases: connection establishment, data exchange and connection release. Consequently, the DLMS/COSEM TCP-based TL provides OSI-style services to the service user(s) for all three phases:

- for the connection establishment phase, the TCP-CONNECT service is provided to the service user TCP connection manager process;
- for the data transfer phase, the TCP-DATA service is provided to the service user DLMS/COSEM AL;
- for the connection closing phase, the TCP-DISCONNECT service is provided to the service user TCP connection manager process;
- in addition, a TCP-ABORT service is provided to the service user DLMS/COSEM AL.

The DLMS/COSEM connection-oriented, TCP-based TL contains the same wrapper sublayer as the DLMS/COSEM UDP-based TL. In addition to transforming OSI-style services to and from TCP function calls, this wrapper provides additional addressing and length information.

The DLMS/COSEM connection-oriented, TCP-based TL is specified in terms of services and protocols. The conversion between OSI-style services and TCP function calls is presented in 7.2.5.

#### 7.2.4.2 Service specification for the DLMS/COSEM TCP-based transport layer

##### 7.2.4.2.1 General

The DLMS/COSEM connection-oriented, TCP-based TL provides the same set of services both at the client and at the server sides, as it is shown in Figure 31.

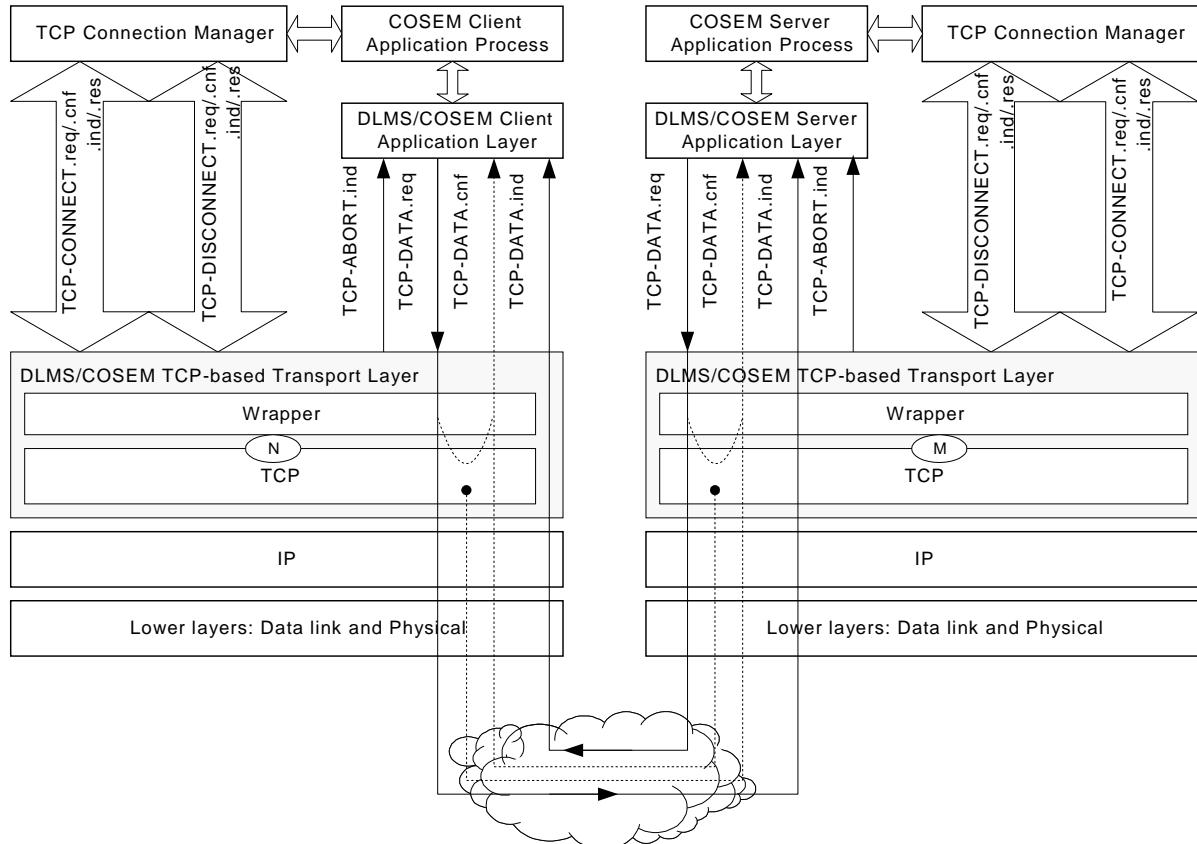
In this communication profile, the full set of the service primitives of the TCP connection management services (TCP-CONNECT and TCP-DISCONNECT) is provided both at the client- and at the server sides. This is to allow the server initiating and releasing a TCP connection, too.

**NOTE** Application association establishment is performed by the client AE.

The service user of the TCP connection management services is not the DLMS/COSEM AL, but the TCP connection manager process. The specification of this process is out of the Scope of this Technical Report. However, the DLMS/COSEM AL sets some requirements concerning this. See in 10.3.4.

An additional COSEM-ABORT service is provided to indicate to the DLMS/COSEM AL the disruption or disconnection of the supporting TCP connection.

Like in the DLMS/COSEM UDP-based TL, the TCP-DATA.confirm service primitive is also optional. However, the TCP-DATA.request service can be confirmed either locally or remotely.



**Figure 31 – Services of the DLMS/COSEM connection-oriented, TCP-based transport layer**

#### 7.2.4.2.2 The TCP-CONNECT service

##### 7.2.4.2.2.1 TCP-CONNECT.request

###### 7.2.4.2.2.1.1 Function

This primitive is the service request primitive for the connection establishment service.

###### 7.2.4.2.2.1.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-CONNECT.request      (
    Local_TCP_Port,
    Remote_TCP_Port,
    Local_IP_Address,
    Remote_IP_Address
)
```

Where:

- Local\_TCP\_Port and Remote\_TCP\_Port identify the local and remote TCP ports respectively.
- The Local\_IP\_Address and Remote\_IP\_Address parameters indicate the IP address of the physical device requesting the TCP connection and of the target physical device, to which the TCP connection requested is to be established.

###### 7.2.4.2.2.1.3 Use

The TCP-CONNECT.request primitive is invoked by the service user TCP connection manager process to establish a connection with the peer DLMS/COSEM TCP-based TL.

##### 7.2.4.2.2.2 TCP-CONNECT.indication

###### 7.2.4.2.2.2.1 Function

This primitive is the service indication primitive for the connection establishment service.

###### 7.2.4.2.2.2.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-CONNECT.indication      (
    Local_TCP_Port,
    Remote_TCP_Port,
    Local_IP_Address,
    Remote_IP_Address
)
```

Where:

- Local\_TCP\_Port and Remote\_TCP\_Port indicate the two TCP ports between which the requested TCP connection is to be established.
- Local\_IP\_Address and Remote\_IP\_Address indicate the IP addresses of the two devices participating in the TCP connection.

###### 7.2.4.2.2.2.3 Use

The TCP-CONNECT.indication primitive is generated by the DLMS/COSEM TCP-based TL following the reception of a TCP packet, indicating to the TCP connection manager process that a remote device is requesting a new TCP connection.

##### 7.2.4.2.2.3 TCP-CONNECT.response

###### 7.2.4.2.2.3.1 Function

This primitive is the service response primitive for the connection establishment service.

###### 7.2.4.2.2.3.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	87/614
-----------------------	------------	-----------------------	--------

```
TCP-CONNECT.response      (
    Local_TCP_Port,
    Remote_TCP_Port,
    Local_IP_Address,
    Remote_IP_Address,
    Result
)
```

Where:

- Local\_TCP\_Port and Remote\_TCP\_Port indicate the two TCP ports between which the connection is being established.
- Local\_IP\_Address and Remote\_IP\_Address indicate the IP addresses of the two physical devices participating in the TCP connection.
- Result indicates that the service user TCP connection manager has accepted the requested TCP connection. Its value is always SUCCESS.

#### 7.2.4.2.2.3.3 Use

The TCP-CONNECT.response primitive is invoked by the TCP connection manager process to indicate to the DLMS/COSEM TCP-based TL whether the TCP connection requested previously has been accepted. The TCP connection manager cannot reject a requested connection.

#### 7.2.4.2.2.4 TCP-CONNECT.confirm

##### 7.2.4.2.2.4.1 Function

This primitive is the service confirm primitive for the connection establishment service.

##### 7.2.4.2.2.4.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-CONNECT.confirm      (
    Local_TCP_Port,
    Remote_TCP_Port,
    Local_IP_Address,
    Remote_IP_Address,
    Result,
    Reason_of_Failure
)
```

Where:

- Local\_TCP\_Port and Remote\_TCP\_Port indicate the two TCP ports between which the connection is being established.
- Local\_IP\_Address and Remote\_IP\_Address indicate the IP addresses of the two physical devices participating in this TCP connection.
- Result indicates, whether the requested TCP connection is established or not. Note that this service primitive is normally the result of a remote confirmation – and as a TCP connection request cannot be rejected, the Result parameter shall always indicate SUCCESS. However, the Result parameter may also indicate FAILURE, when it is locally confirmed. In this case the Reason\_of\_Failure parameter indicates the reason for the failure.

#### 7.2.4.2.2.4.3 Use

The TCP-CONNECT.confirm primitive is generated by the DLMS/COSEM TCP-based TL to indicate to the service user TCP connection manager process the result of a TCP-CONNECT.request service invocation received previously.

#### 7.2.4.2.3 The TCP-DISCONNECT service

##### 7.2.4.2.3.1 TCP-DISCONNECT.request

##### 7.2.4.2.3.1.1 Function

This primitive is the service request primitive for the connection termination service.

88/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
--------	------------	-----------------------	-----------------------

#### 7.2.4.2.3.1.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-DISCONNECT.request ( Local_TCP_Port,
                           Remote_TCP_Port,
                           Local_IP_Address,
                           Remote_IP_Address )
```

Where:

- The service parameters are the identifiers of the TCP connection to be released.
- Local\_TCP\_Port and Local\_IP\_Address designate the local TCP port and IP address of the requesting device and application.
- Remote\_IP\_Address and Remote\_TCP\_Port refer to the remote device and application.

#### 7.2.4.2.3.1.3 Use

The TCP-DISCONNECT.request primitive is invoked by the service user TCP connection manager process to request the disconnection of an existing TCP connection.

#### 7.2.4.2.3.2 TCP-DISCONNECT.indication

##### 7.2.4.2.3.2.1 Function

This primitive is the service indication primitive for the connection termination service.

##### 7.2.4.2.3.2.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-DISCONNECT.indication ( Local_TCP_Port,
                             Remote_TCP_Port,
                             Local_IP_Address,
                             Remote_IP_Address,
                             Reason )
```

Where:

- Local\_TCP\_Port, Remote\_TCP\_Port, Local\_IP\_Address, Remote\_IP\_Address identify the TCP connection, which is either requested to be released by the peer device, or has been aborted.
- The Reason parameter indicates whether the service is invoked because of the peer device has requested a TCP disconnection (Reason == REMOTE\_REQ), or it is locally originated by detecting a kind of event, which implies the disconnection of the TCP connection (Reason == ABORT).

NOTE The DLMS/COSEM transport layer may give more detailed information about the reason for the ABORT via layer management services. However, layer management services are out of the scope of this Technical Report.

##### 7.2.4.2.3.2.3 Use

The TCP-DISCONNECT.indication primitive is generated by the DLMS/COSEM TCP-based TL to the service user TCP connection manager process to indicate that the peer entity has requested the disconnection of an existing TCP connection. The same primitive is used also to indicate if the TL detects a non-solicited disconnection of an existing TCP connection (for example, when the physical connection breaks down).

#### 7.2.4.2.3.3 TCP-DISCONNECT.response

##### 7.2.4.2.3.3.1 Function

This primitive is the service response primitive for the connection termination service.

##### 7.2.4.2.3.3.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	89/614
-----------------------	------------	-----------------------	--------

```
TCP-DISCONNECT.response (
    Local_TCP_Port,
    Remote_TCP_Port,
    Local_IP_Address,
    Remote_IP_Address,
    Result
)
```

Where:

- Local\_TCP\_Port and Remote\_TCP\_Port identify the two TCP ports between which the TCP connection has to be disconnected.
- Local\_IP\_Address and Remote\_IP\_Address indicate the IP addresses of the two physical devices participating in the TCP connection to be disconnected.
- Result indicates that the service user TCP connection manager process has accepted to disconnect the TCP connection referenced. The value of this parameter is always SUCCESS.

#### 7.2.4.2.3.3.3 Use

The TCP-DISCONNECT.response primitive is invoked by the TCP connection manager process to indicate to the DLMS/COSEM TCP-based TL whether the previously requested TCP disconnection is accepted. Note, that the TCP connection manager process cannot reject the requested disconnection. This service primitive is invoked only if the corresponding TCP-DISCONNECT.indication service indicated a remotely initiated disconnection request (Reason == REMOTE\_REQ).

#### 7.2.4.2.3.4 TCP-DISCONNECT.confirm

##### 7.2.4.2.3.4.1 Function

This primitive is the service confirm primitive for the connection termination service.

##### 7.2.4.2.3.4.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-DISCONNECT.confirm (
    Local_TCP_Port,
    Remote_TCP_Port,
    Local_IP_Address,
    Remote_IP_Address,
    Result,
    Reason_of_Failure
)
```

Where:

- Local\_TCP\_Port and Remote\_TCP\_Port identify the two TCP ports between which the TCP connection has to be disconnected.
- Local\_IP\_Address and Remote\_IP\_Address indicate the IP addresses of the two physical devices participating in the TCP connection to be disconnected.
- Result indicates, whether the disconnection of the TCP connection referenced has succeeded or not. Normally, this service primitive is invoked as the result of a remote confirmation, and as a TCP disconnection request cannot be rejected, the value of the Result parameter is always SUCCESS. However, the Result parameter may also indicate FAILURE, when it is locally confirmed. In this case the Reason\_of\_Failure parameter indicates the reason of the failure.

##### 7.2.4.2.3.4.3 Use

The TCP-DISCONNECT.confirm primitive is invoked by the DLMS/COSEM TCP-based TL to confirm to the service user TCP connection manager the result of a previous TCP-DISCONNECT.request service invocation.

#### 7.2.4.2.4 The TCP-ABORT service

##### 7.2.4.2.4.1 TCP-ABORT.indication

###### 7.2.4.2.4.1.1 Function

This primitive is the service indication primitive for the connection termination service.

###### 7.2.4.2.4.1.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-ABORT.indication      (
    Local_TCP_Port,
    Remote_TCP_Port,
    Local_IP_Address,
    Remote_IP_Address,
    Reason
)
```

Where:

- Local\_TCP\_Port and Remote\_TCP\_Port identify the two TCP ports the connection between which has aborted.
- Local\_IP\_Address and Remote\_IP\_Address indicate the IP addresses of the two physical devices having been participated in the TCP connection aborted.
- Reason indicates the reason of the TCP abort. This parameter is optional.

###### 7.2.4.2.4.1.3 Use

The TCP-ABORT.indication primitive is generated by the DLMS/COSEM TCP-based TL to indicate to the service user DLMS/COSEM AL a non-solicited disruption of the supporting TCP connection.

When this indication is received, the DLMS/COSEM AL shall release all AAs established using this TCP connection, and shall indicate this to the COSEM AP using the COSEM-ABORT.indication service primitive. See also 9.3.4.

#### 7.2.4.2.5 The TCP-DATA service

##### 7.2.4.2.5.1 TCP-DATA.request

###### 7.2.4.2.5.1.1 Function

This primitive is the service request primitive for the connection mode data transfer service.

###### 7.2.4.2.5.1.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-DATA.request      (
    Local_wPort,
    Remote_wPort,
    Local_TCP_Port,
    Remote_TCP_Port,
    Local_IP_Address,
    Remote_IP_Address,
    Data_Length,
    Data
)
```

Where:

- The Local\_wPort, Local\_TCP\_Port and Local\_IP\_Address parameters indicate wrapper Port number, TCP Port number and IP address parameters of the device / DLMS/COSEM AE requesting to send the Data. The Remote\_wPort, Remote\_TCP\_Port and Remote\_IP\_Address parameters indicate the wrapper Port number, TCP Port number and IP address parameters belonging to the device / DLMS/COSEM AE to which the Data is to be transmitted.

- Data\_Length indicates the length of the Data parameter in bytes.
- Data contains the xDLMS APDU to be transferred to the peer AL.

#### 7.2.4.2.5.1.3 Use

The TCP-DATA.request primitive is invoked by either the client or the server DLMS/COSEM AL to request sending an APDU to a single peer application.

The reception of this primitive shall cause the wrapper sublayer to pre-fix the wrapper-specific fields (Local\_wPort, Remote\_wPort and the Data\_Length) to the xDLMS APDU received, and then to call the SEND() function of the TCP sublayer with the properly formed WPDU, see at 7.2.3.3.2, as DATA. The TCP sublayer shall transmit the WPDU to the peer TCP sublayer as described in STD0007.

#### 7.2.4.2.5.2 TCP-DATA.indication

##### 7.2.4.2.5.2.1 Function

This primitive is the service indication primitive for the connection mode data transfer service.

##### 7.2.4.2.5.2.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-DATA.indication ( Local_wPort,  
                         Remote_wPort,  
                         Local_TCP_Port,  
                         Remote_TCP_Port,  
                         Local_IP_Address,  
                         Remote_IP_Address,  
                         Data_Length,  
                         Data  
                     )
```

Where:

- The Local\_wPort, Local\_TCP\_Port and Local\_IP\_Address parameters indicate wrapper Port number,
- TCP Port number and IP address parameters belonging to the device / DLMS/COSEM AE receiving the Data.
- The Remote\_wPort, Remote\_TCP\_Port and Remote\_IP\_Address parameters indicate the wrapper Port number, TCP Port number and IP address parameters belonging to the device / DLMS/COSEM AE which has sent the Data.
- Data\_Length indicates the length of the Data parameter in bytes.
- Data contains the xDLMS APDU received from the peer AL.

##### 7.2.4.2.5.2.3 Use

The TCP-DATA.indication primitive is generated by the DLMS/COSEM TL to indicate to the service user DLMS/COSEM AL that an xDLMS APDU has been received from a remote device. It is generated following the reception of a complete APDU (in one or more TCP packets) by the DLMS/COSEM TCP-based TL, if both the Local\_TCP\_Port and Local\_wPort parameters in the TCP packet(s) carrying the APDU contain valid port numbers, meaning that there is a DLMS/COSEM AE in the receiving device bound to the given port numbers. Otherwise, the message received shall simply be discarded.

#### 7.2.4.2.5.3 TCP-DATA.confirm

##### 7.2.4.2.5.3.1 Function

This primitive is the optional service confirm primitive for the connection mode data transfer service.

##### 7.2.4.2.5.3.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-DATA.confirm ( Local_wPort,
```

```

        Remote_wPort,
        Local_TCP_Port,
        Remote_TCP_Port,
        Local_IP_Address,
        Remote_IP_Address,
        Confirmation_Type,
        Result
    )

```

Where:

- The Local\_wPort, Remote\_wPort, Local\_TCP\_Port, Remote\_TCP\_Port, Local\_IP\_Address and Remote\_IP\_Address parameters carry the same values as the corresponding TCP-DATA.request service being confirmed.
- Confirmation\_Type indicates whether the confirmation service is a LOCAL or a REMOTE confirmation.
- Result indicates the result of the previous TCP-DATA.request service. Its value is either OK or NOK, but the meaning of this depends on the implementation of the .confirm primitive. See 7.2.4.3.5.4.

#### 7.2.4.2.5.3.3 Use

The TCP-DATA.confirm primitive is optional. If implemented, it is generated by the DLMS/COSEM TL to confirm to the service user DLMS/COSEM AL the result of the execution of the previous .request primitive.

### 7.2.4.3 Protocol specification for the DLMS/COSEM TCP-based transport layer

#### 7.2.4.3.1 General

As it is shown in Figure 27, the DLMS/COSEM CO, TCP-based TL includes the Internet standard TCP layer as specified in STD0007, and the DLMS/COSEM-specific wrapper sublayer.

In the TCP-based TL the wrapper sublayer is more complex than in the UDP-based TL. On the one hand – similarly to the UDP-based TL – its main role is also to ensure source and destination DLMS/COSEM AE identification using the wPort numbers, and to convert OSI-style TCP-DATA service primitives to and from the SEND() and RECEIVE() interface functions provided by the standard TCP. On the other hand, the wrapper sublayer in the TCP-based TL has also the task to help the service user DLMS/COSEM ALs to exchange complete APDUs.

TCP is a “streaming” protocol meaning that it does not preserve data boundaries. Without entering into the details here (see more in 7.2.5.4) this means, that the SEND() and RECEIVE() function calls of the TCP sublayer return with success even if the number of the bytes sent / received actually is less than the number of bytes requested to be sent / received. It is the responsibility of the wrapper sublayer to know how much data had to be sent / received, to keep track how much has been actually sent / received, and repeat the operation until the complete APDU is transmitted.

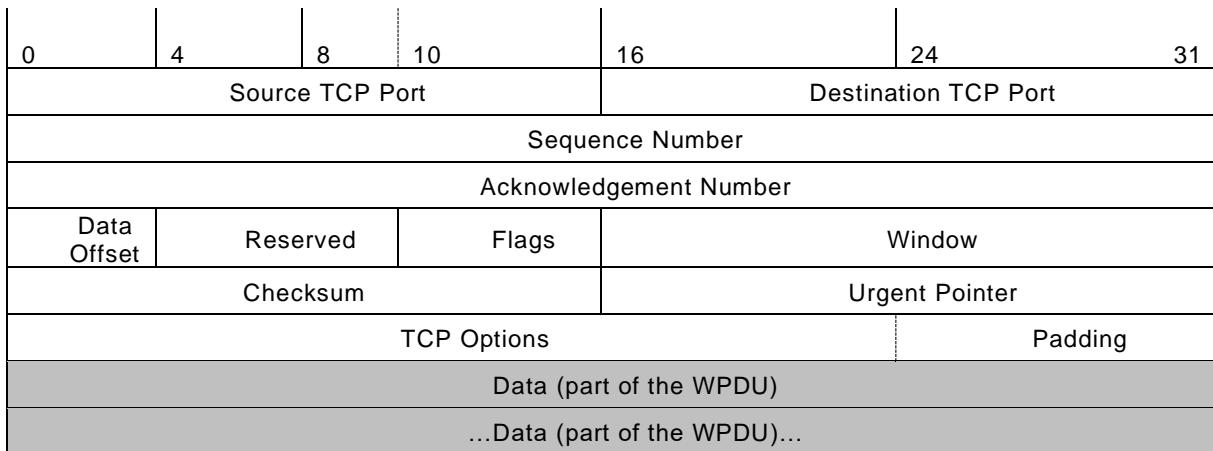
Consequently, the wrapper sublayer in the TCP-based DLMS/COSEM TL is not a state-less entity: it is doing the above described track-keeping – re-trying procedure in order to make the “streaming” nature of the TCP transparent to the service user DLMS/COSEM AL.

#### 7.2.4.3.2 The wrapper protocol data unit (WPDU)

The wrapper protocol data unit is as it is specified at 7.2.3.3.2.

#### 7.2.4.3.3 The DLMS/COSEM TCP-based transport layer protocol data unit

WPDUs are transmitted in one or more TCP packets. The TCP Packet is specified in STD0007, and encapsulates a part of the WPDU in its Data Field, as it is shown in Figure 32. The reason for having only a part of the WPDU in a TCP packet is the “streaming” nature of the TCP already mentioned.

**Figure 32 – The TCP packet format**

From the external point of view the DLMS/COSEM TCP-based TL PDU is an ordinary TCP packet: any COSEM specific element, including the wrapper-specific header in the first TCP packet, is inside the packet's Data field.

The source and destination TCP ports may refer to either local or remote TCP ports, depending on the direction of the data transfer (i.e. from the point of view of the Sender device the source TCP port in a TCP Packet corresponds to the Local\_TCP\_Port, but from the point of view of the Receiver device, the source TCP port of a Datagram corresponds to the Remote\_TCP\_Port service parameter).

#### 7.2.4.3.4 Reserved wrapper port numbers

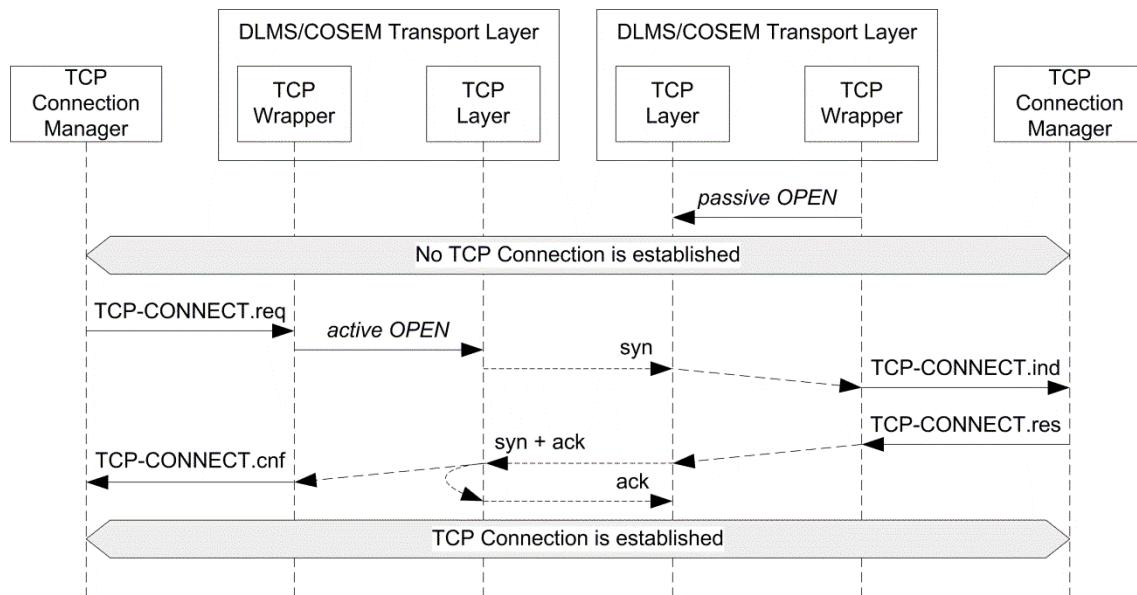
Reserved wPort Numbers are specified in Table 2.

#### 7.2.4.3.5 Definition of the procedures

##### 7.2.4.3.5.1 TCP connection

Establishment of a TCP connection is initiated by the TCP-CONNECT.request service invocation. Although this service – as all DLMS/COSEM TL services – is provided to the service user entity by the wrapper sublayer, the TCP connection is established between the two (local and remote) TCP sublayers. The role of the wrapper in this procedure is just to convert the TCP-CONNECT service primitives (.request, .indication, .response and .confirm) to and from TCP function calls.

From the service user point of view, only the TCP-CONNECT service primitives are visible: according to this, the TCP connection establishment takes place as it is shown in Figure 33.

**Figure 33 – TCP connection establishment**

The TCP connection is established using a three-way handshake mechanism, as described in STD0007. This requires three message exchanges as shown above and guarantees that both sides know that the other side is ready to transmit and also that the two sides are synchronized: the initial sequence numbers are agreed upon.

Both the client and server side TCP connection manager processes are allowed to initiate the TCP connection. To establish the connection, one of them plays the role of the initiator, and the other that of the responder.

In order to be able to respond, the responder has to perform a ‘passive’ opening before receiving the first, SYN packet. To do this, it has to contact the local operating system (OS) to indicate, that it is ready to accept incoming connection requests. As the result of this contact, the OS assigns a TCP port number to that end-point of the connection and reserves the resources required for a future connection – but no message is sent out.

**NOTE** In the case of the DLMS/COSEM transport layer, the implementation forces the OS to assign the requested TCP / UDP port number to the local end point of the connection.

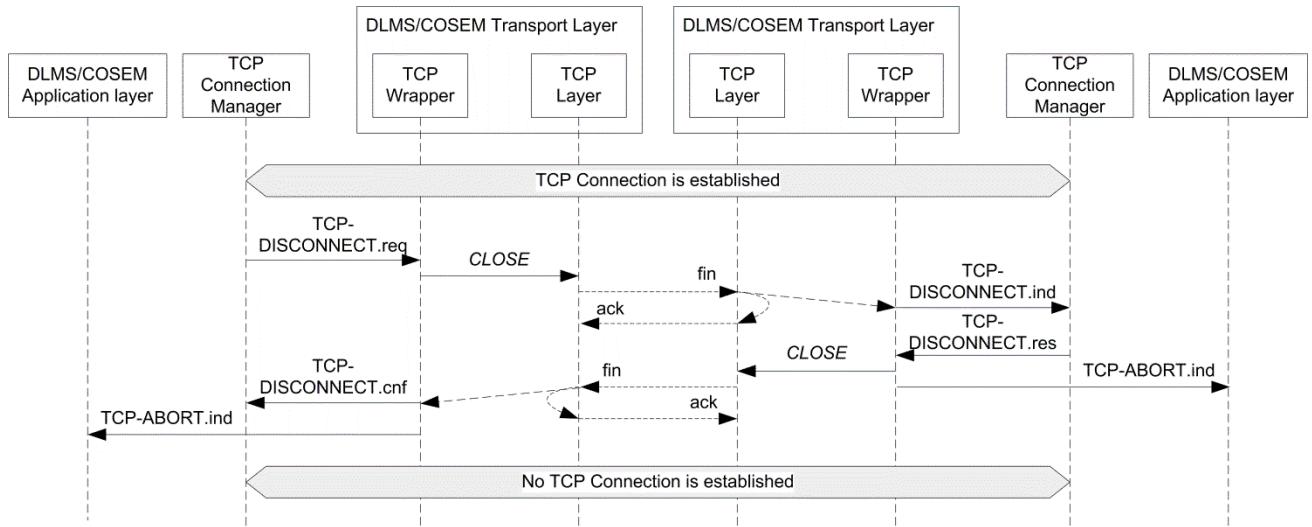
In the case of the DLMS/COSEM TCP-based TL, the wrapper sublayer initiates this passive opening autonomously during system initialisation. In other words, as this passive opening is the responsibility of the wrapper sublayer, no service is provided to an external entity to initiate the passive opening.

As both the client and the server side TCP connection manager processes are allowed to play the role of the “Responder” application, the TLs on both sides shall perform a passive opening during the system initialisation.

More details about TCP connection establishment are provided in 7.2.5.1.

#### 7.2.4.3.5.2 TCP disconnection

The TCP is disconnected using the TCP-DISCONNECT service, as shown in Figure 34.

**Figure 34 – TCP disconnection**

The procedure can be initiated either by the client or the server side TCP connection manager process, by invoking the TCP-DISCONNECT.request primitive. This request is transformed by the “wrapper” to a CLOSE () function call to the TCP interface.

The TCP sends a fin segment, which is acknowledged by the peer TCP.

**NOTE** TCP uses an improved 3-way handshake to release a connection, to ensure that possible duplication and delay – introduced by the non-reliable IP layer – do not pose problems. More about this procedure can be found in STD0007.

At the same time, through the wrapper, the TCP-DISCONNECT.indication primitive is generated, informing the user TCP connection manager that the connection is closing. The connection manager – in order to gracefully release the connection – responds with a TCP-DISCONNECT.response primitive. The TCP wrapper calls the CLOSE function and the TCP sends out its fin segment. At the same time, the TCP wrapper indicates the closing of the TCP connection to the DLMS/COSEM AL using the TCP-ABORT.indication primitive.

On the requesting side, the TCP sends an acknowledgement and upon the reception of this by the peer the TCP connection is deleted. At the same time, the wrapper generates the TCP-DISCONNECT.confirm primitive informing the connection manager process that the disconnection request has been accepted. Similarly to the peer, the TCP disconnection is also indicated to the DLMS/COSEM AL with the help of the COSEM-ABORT.indication primitive.

More details about TCP disconnection are provided in 7.2.5.2.

#### 7.2.4.3.5.3 TCP connection abort

The DLMS/COSEM TCP-based TL indicates the disruption or disconnection of the supporting TCP connection to the DLMS/COSEM AL with the help of the TCP-ABORT.indication primitive. Note, that this is the only TCP connection management service provided to the DLMS/COSEM AL.

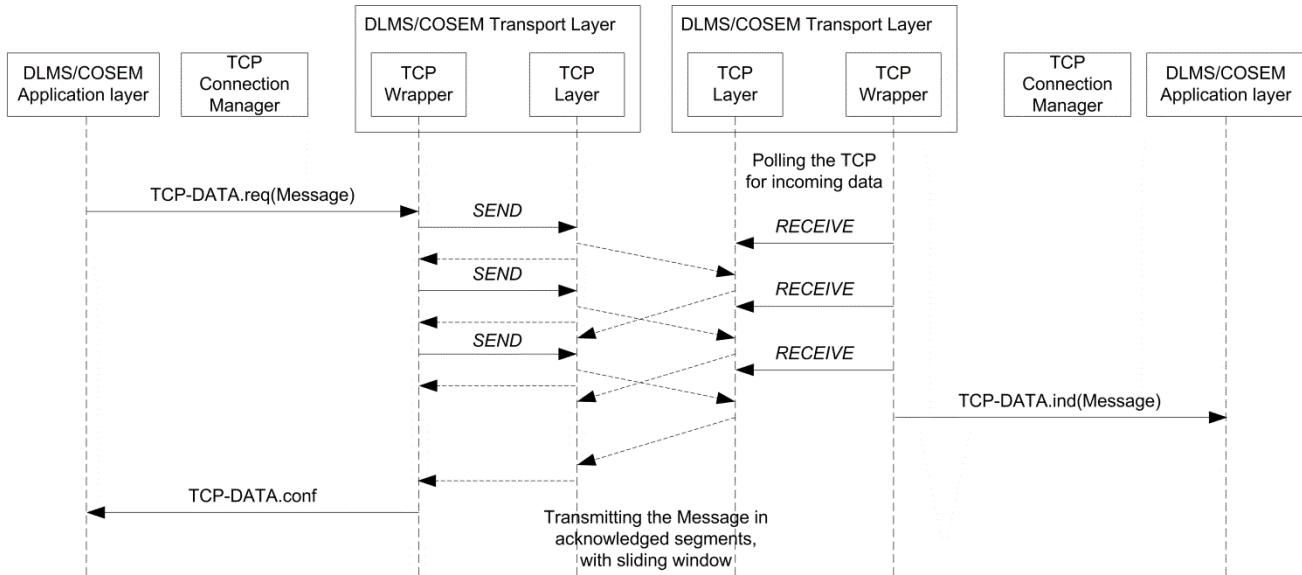
The service is invoked either when the TCP connection is disconnected by the TCP connection manager process – the case of graceful disconnection – or when the TCP disconnection occurs in a non-solicited manner, for example the TCP sublayer is detecting a non-resolvable error or the physical connection is shut down.

The purpose of this service is to inform the DLMS/COSEM AL about the disruption of the TCP connection, so that it could release all existing AAs.

#### 7.2.4.3.5.4 Data transfer using the TCP-DATA service

Once the TCP connection is established, reliable data transfer can be performed via this connection. Although providing this reliable data transfer is a quite complex operation involving reliability mechanisms such as *Positive Acknowledgement with Retransmission* (PAR) or flow control with sliding windows – provided by TCP and specified in STD0007 – the DLMS/COSEM TCP-based TL layer

provides only data transfer service, the TCP-DATA service, as shown in Figure 35. The use of the TCP-DATA service is the same both on the client and at the server side.



**Figure 35 – Data transfer using the COSEM TCP-based transport layer**

The optional TCP-DATA.confirm primitive indicates the result of the TCP-DATA.request primitive invoked previously, which is either OK or NOK. However, the meaning of this result is implementation dependent. When the .confirm primitive is implemented as a local confirmation, the result indicates whether the DLMS/COSEM TL was able to buffer for sending or to send out the APDU or not. When it is implemented as a remote confirmation, the result indicates whether the APDU has been successfully delivered to the destination or not.

As shown in Figure 35, the message (a WPDU) may be transported (sent / received) in more than one TCP packet. It is because TCP sends data as a stream of octets, without preserving data boundaries. It is the responsibility of the wrapper sublayer to hide this property of the TCP sublayer from the service user DLMS/COSEM AL. The sender side wrapper keeps track about the amount of data sent with one SEND() function call and repeats the operation until the whole WPDU is sent. The receiver side wrapper continues to receive incoming TCP packets until a complete WPDU is received. For more details, see 7.2.5.4.

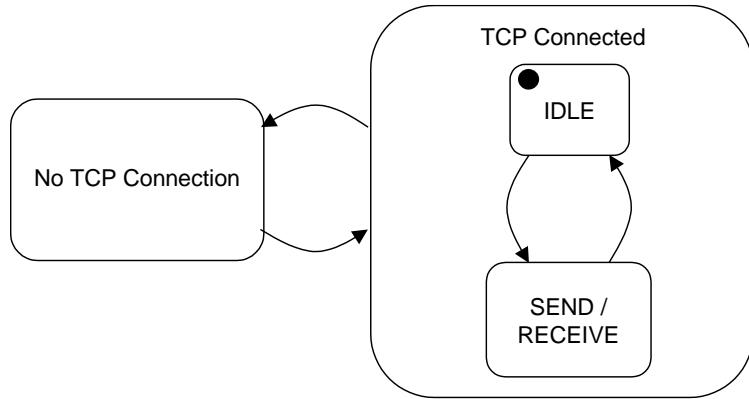
#### 7.2.4.3.5.5 High-level state transition diagram of the wrapper sublayer

The high level state-diagram of the wrapper sublayer is shown in Figure 36.

In both macro-states – No TCP Connection and TCP Connected – the wrapper keeps polling the TCP layer for its connection status, and transits into the other macro-state if the status has changed.

The wrapper enters always into the IDLE sub-state of the TCP Connected state, and transits to the composite SEND/RECEIVE state either on a TCP-DATA.request or on the reception of a TCP packet. In this state, the wrapper sends and/or receives WPDUs, as described in 7.2.5.

NOTE TCP on the top of a full-duplex lower layer protocol stack may simultaneously send and receive.



**Figure 36 – High-level state transition diagram for the wrapper sublayer**

### 7.2.5 Converting OSI-style TL services to and from RFC-style TCP function calls

#### 7.2.5.1 Transport layer and TCP connection establishment

As specified in STD0007, a TCP connection is established by calling the OPEN function. This function can be called in *active* or *passive* manner.

According to the TCP connection state diagram (Figure 37) a *passive* OPEN takes the caller device to the LISTEN state, waiting for a connection request from any remote TCP and port.

An *active* OPEN call makes the TCP to establish the connection to a remote TCP.

The establishment of a TCP Connection is performed by using the so-called “Three-way handshake” procedure. This is initiated by one TCP calling an *active* OPEN and responded by another TCP, the one, which has already been called a *passive* OPEN and consequently is in the LISTEN state.

The message sequence – and the state transitions corresponding to that message exchange – for this “three-way handshake” procedure are shown in Figure 38.

This process, consisting of three messages, establishes the TCP connection and “synchronizes” the initial sequence numbers at both sides. This mechanism has been carefully designed to guarantee, that both sides are ready to transmit data and know that the other side is ready to transmit as well. Note that the procedure also works if two TCPs simultaneously initiate the procedure.

**NOTE** Sequence numbers are part of the TCP packet, and are fundamental to reliable data transfer. For more details about sequence numbers (or other TCP related issues), please refer to STD0007.

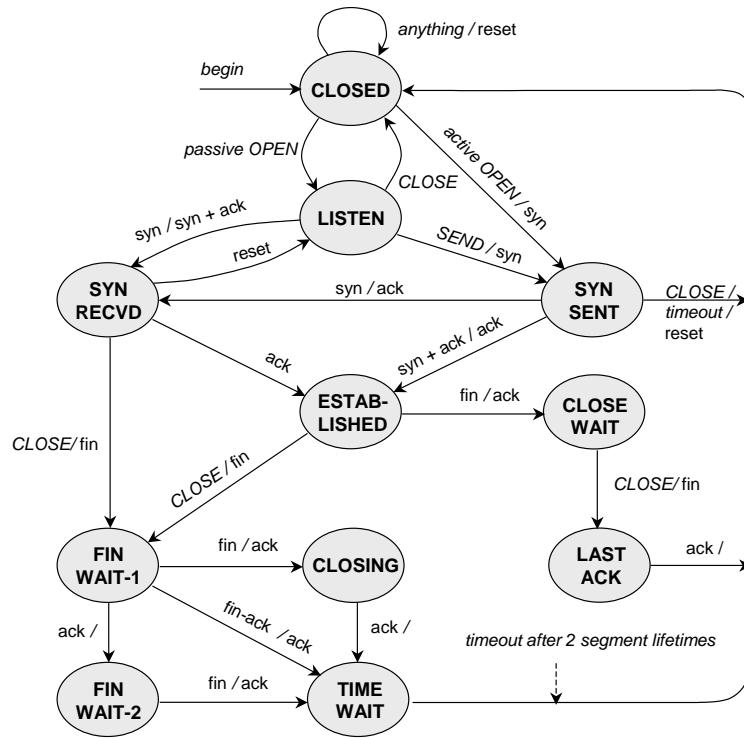
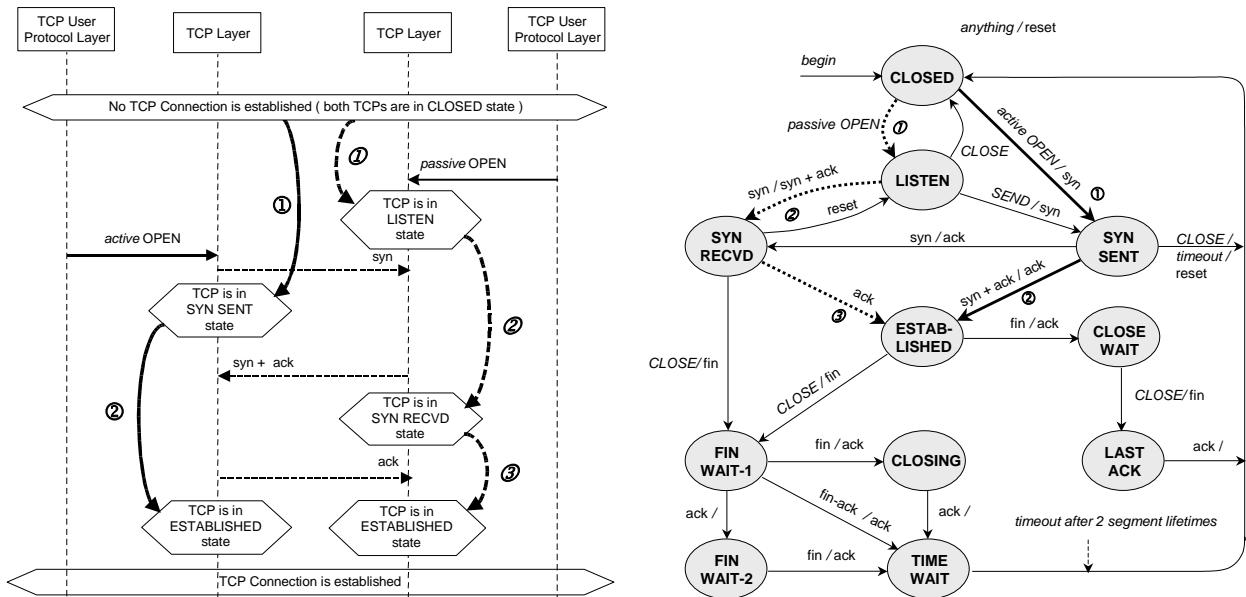


Figure 37 – TCP connection state diagram



NOTE In the case of the DLMS/COSEM transport layer, the TCP user protocol layer is the wrapper sublayer.

Figure 38 – MSC and state transitions for establishing a transport layer and TCP connection

### 7.2.5.2 Closing a transport layer and a TCP connection

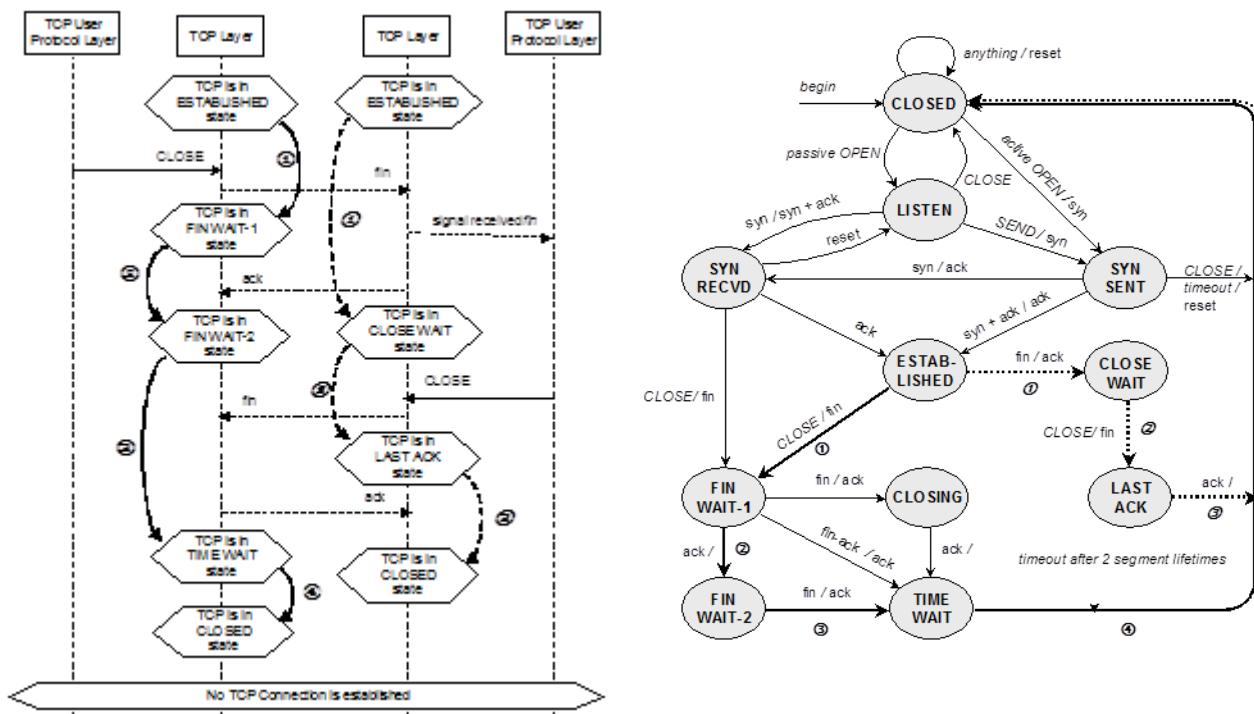
Closing a TCP connection is done by calling the CLOSE function, generally when there is no more data to be sent.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	99/614
-----------------------	------------	-----------------------	--------

Upon the invocation of the TCP-DISCONNECT.request service primitive by the TCP connection manager process, the wrapper sublayer invokes the CLOSE function of the TCP sublayer.

However, as the TCP connection is full duplex, the other side may still have data to send. Therefore, after calling the CLOSE function, the TCP-based transport later may continue to receive data and send it to the DLMS/COSEM AL, until it is told that the other side has CLOSED, too. At this point it generates the COSEM-ABORT.indication primitive, and all AAs are released.

The message sequence chart and the state transitions corresponding to a successful TCP connection release are shown in Figure 39.

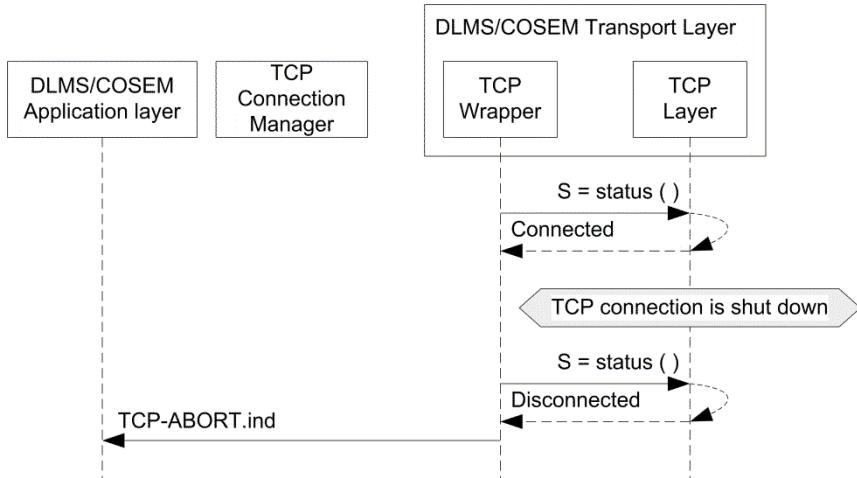


**NOTE** In the case of the DLMS/COSEM TL, the TCP user protocol layer is the wrapper sublayer.

**Figure 39 – MSC and state transitions for closing a transport layer and TCP connection**

### 7.2.5.3 TCP connection abort

STD0007 does not specify a standard function to indicate an unexpected abort at TCP level. However, it can be detected by the TCP user entity by polling the status of the TCP with the STATUS() function, as shown in Figure 40.

**Figure 40 – Polling the TCP sublayer for TCP abort indication**

#### 7.2.5.4 Data transfer using the TCP-DATA service

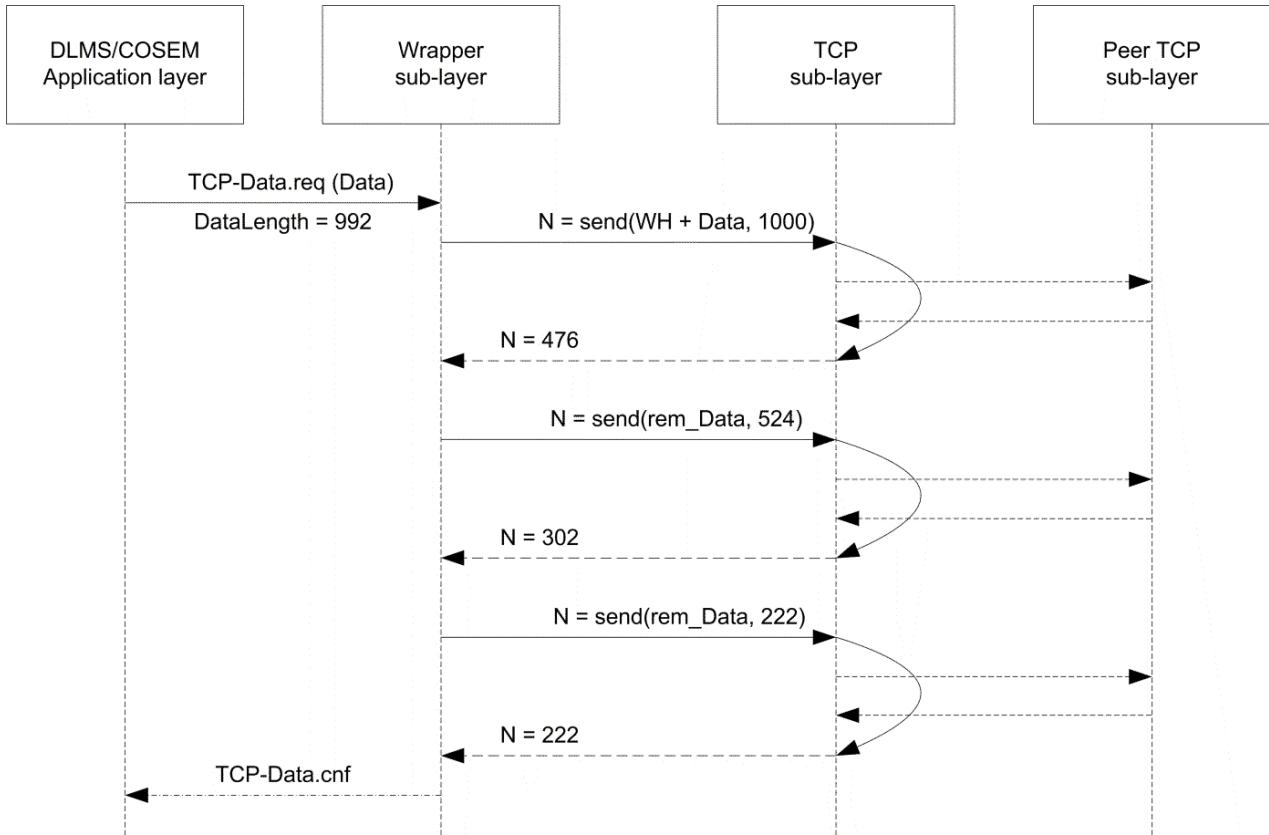
To send an APDU to the peer, the DLMS/COSEM AL simply invokes the TCP-DATA.request primitive of the DLMS/COSEM TCP-based TL. Also, when a complete APDU is received, this is indicated to the DLMS/COSEM AL with the help of the TCP-DATA.indication primitive. Thus, for the AL the TL behaves as if it would transport the whole APDU in one piece.

However, as TCP is a streaming protocol, not preserving data boundaries – as described in 7.2.4.3.1 – nothing ensures that an APDU is actually transmitted in one TCP packet. As already mentioned in 7.2.4.3.5.4, in the DLMS/COSEM TCP-based TL it is the responsibility of the wrapper sublayer to “hide” the streaming nature of the TCP sublayer.

The following example illustrates how the wrapper sublayer accomplishes this task. Let's suppose that an AL entity wants to send an APDU containing 992 bytes via the DLMS/COSEM TCP-based TL.

**NOTE** Both the client- and server side ALs can be either sender or receiver.

It invokes the TCP-DATA.request service with this APDU as the DATA service parameter as shown in Figure 41.



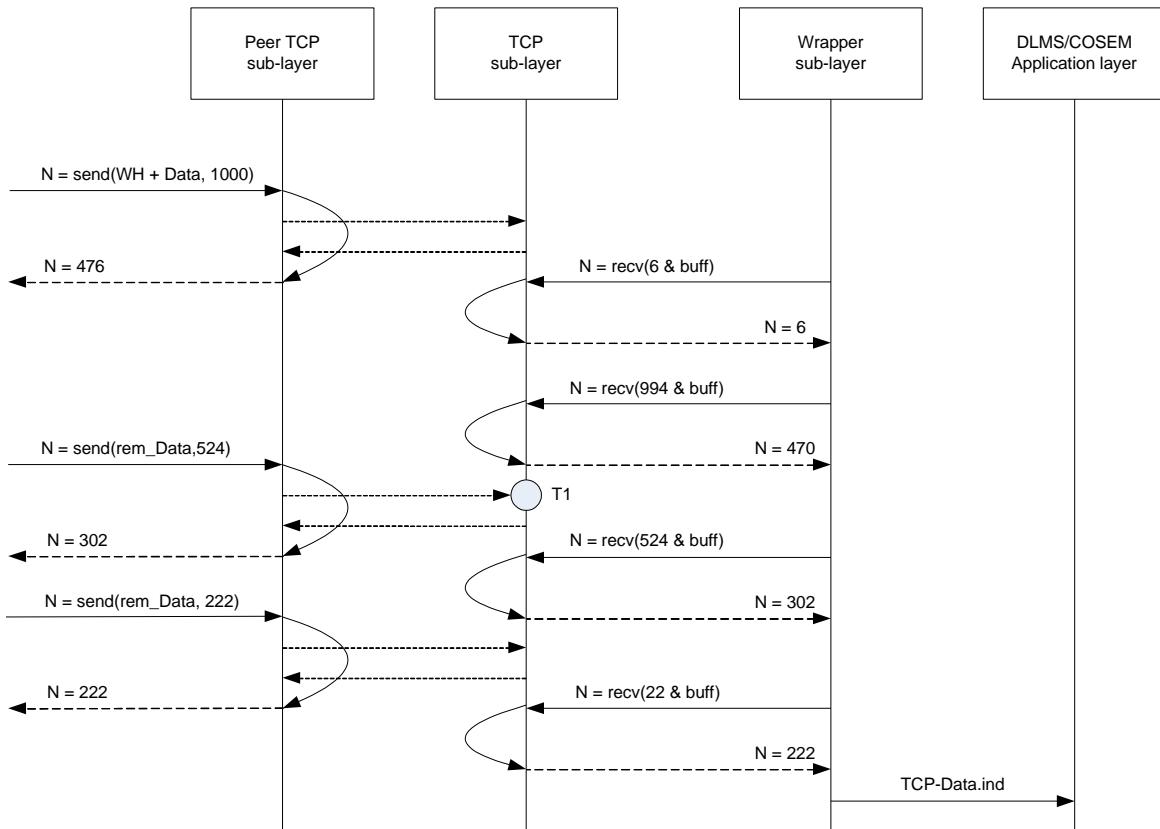
**Figure 41 – Sending an APDU in three TCP packets**

Upon the reception of this service invocation, the wrapper sublayer constructs the WPDU: it pre-fixes the APDU with the wrapper header (WH), including the local and remote wPort numbers and the APDU length. It calls then the SEND() function of the TCP sublayer, requesting to send the WPDU, which is now 1000 bytes long: 8 bytes of wrapper header plus 992 bytes of APDU.

The SEND() function returns with the number of bytes sent or an error (a negative value). Let's suppose that no error occurs and the SEND() function successfully returns with the value 476. This number is the number of bytes sent. This also illustrates the meaning of the “streaming” nature of the TCP: in fact, the SEND() function returns with success even if the number of bytes sent is less than the number of bytes requested to be sent. From the value returned, the wrapper knows that not the whole WPDU has been sent. It calls the SEND() function again, with the remaining part of the WPDU – and so on, until the complete WPDU is sent.

As already mentioned in 7.2.4.3.5.4, depending on the implementation, the successful return of the SEND() function may even not mean that something has been really sent to the network. It may mean only that the protocol implementation took and buffered the data. It may happen that the protocol implementation delays the transmission to comply with protocol conventions or network traffic related algorithms.

On the receiving side, it is also the responsibility of the wrapper sublayer to assemble the complete APDU before invoking the TCP-DATA.indication primitive. This is possible by using the length bytes of the WPDU header. The wrapper repeats RECEIVE() calls until the number of bytes, indicated in the WPDU header is received. This is shown in Figure 42.



NOTE 1 As calling the RECEIVE() function is asynchronous with regard to the TCP communications, it is perfectly possible, that the receiver calls the RECEIVE() function at a moment, when the reception of a TCP packet is in progress (T1 on the Figure above) – or even if when no characters have been received since the last RECEIVE() call. It does not lead to erroneous reception: it increases only the number of necessary RECEIVE() function calls to get the complete message.

NOTE 2 It is also possible that one or more SEND() calls result in sending more than one TCP packet. It does not lead to erroneous reception either: sooner or later, the receiver gets the whole message.

**Figure 42 – Receiving the message in several packets**

All these SEND() and RECEIVE() calls are internal to the DLMS/COSEM TL. The service user DLMS/COSEM AL simply uses the TCP-DATA services, and observes a reliable data transfer service preserving the data boundaries of the APDUs.

### 7.3 The DLMS/COSEM CoAP based transport layer

#### 7.3.1 Scope

This subclause 7.3 specifies the connection-less DLMS/COSEM CoAP transport layer (DLMS/COSEM CoAP TL) for the DLMS/COSEM CoAP based communication profile (DLMS/COSEM\_on\_CoAP) for exchanging DLMS/COSEM application messages over IP based (or capable) networks.

#### 7.3.2 Overview

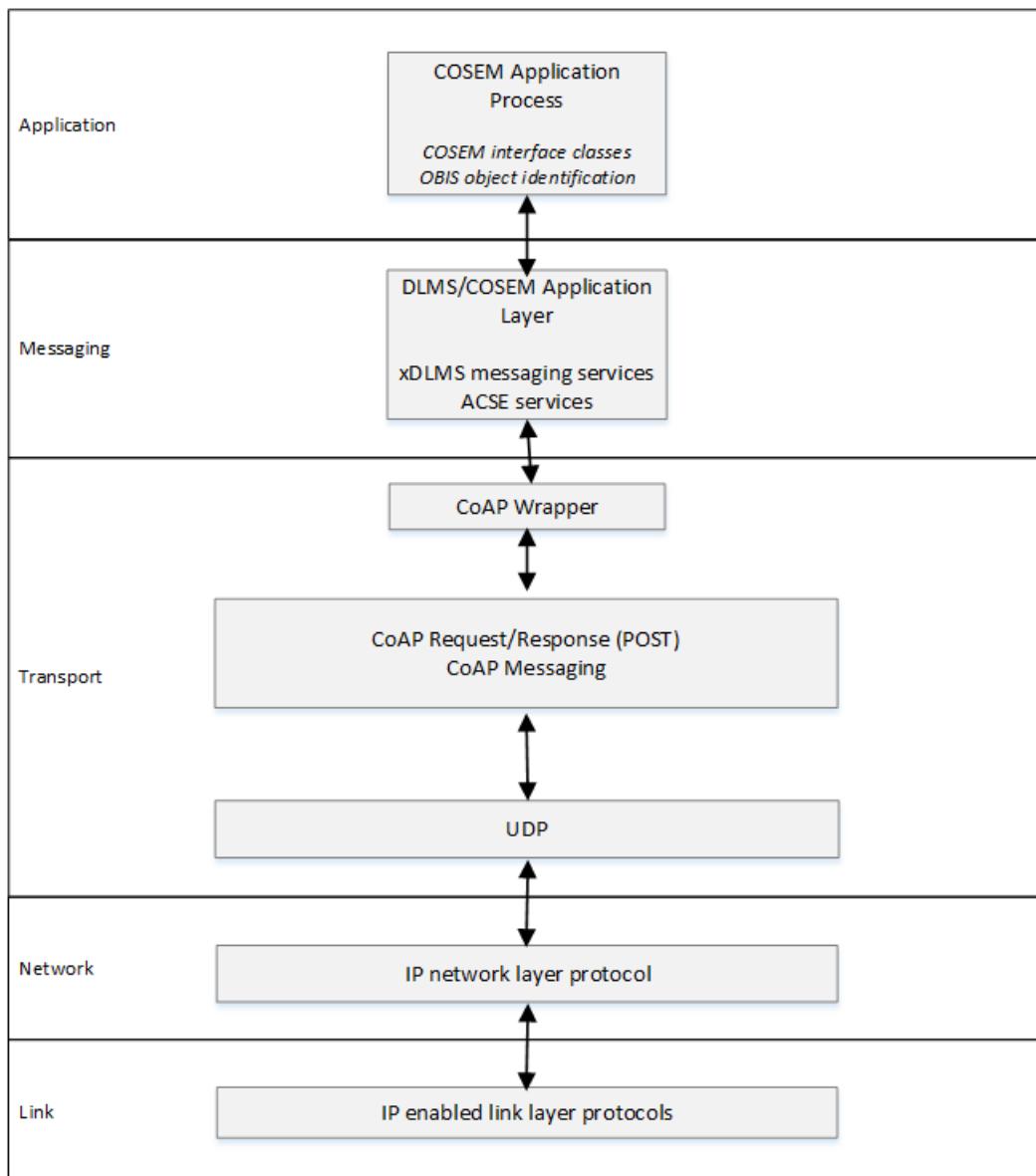
The Constrained Application Protocol (CoAP) is a specialized Internet Application Protocol defined by the IETF Core working group. CoAP is designed particularly for use in resource-constrained devices for communication over non-constrained or constrained internet communication networks (e.g., low-power, lossy networks). CoAP is designed to provide efficient data transport capabilities while also meeting specialized requirements such as reliability, multicast support, very low overhead, efficiency, and simplicity.

The CoAP based DLMS/COSEM CoAP TL provides both unreliable and reliable transport services. The unreliable CoAP service supports multicasting and broadcasting. The DLMS/COSEM CoAP TL provides OSI-style services to the service user DLMS/COSEM AL.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	103/614
-----------------------	------------	-----------------------	---------

The DLMS/COSEM CoAP TL consists of a CoAP request/response layer and its underlying CoAP messaging layer as specified in RFC 7252. The DLMS/COSEM CoAP TL embeds the CoAP block transfer mechanism as specified by RFC 7959. In addition, it includes a DLMS/COSEM CoAP wrapper layer that provides the OSI-style DLMS/COSEM CoAP-DATA service primitives to the DLMS/COSEM AL. The DLMS/COSEM CoAP TL embeds a standard UDP transport layer running on top of an IPv4 or IPv6 network layer.

When the DLMS/COSEM AL uses the DLMS/COSEM CoAP TL for communication, the DLMS/COSEM AL operates as an Internet standard serialization and security standard. See Figure 43.

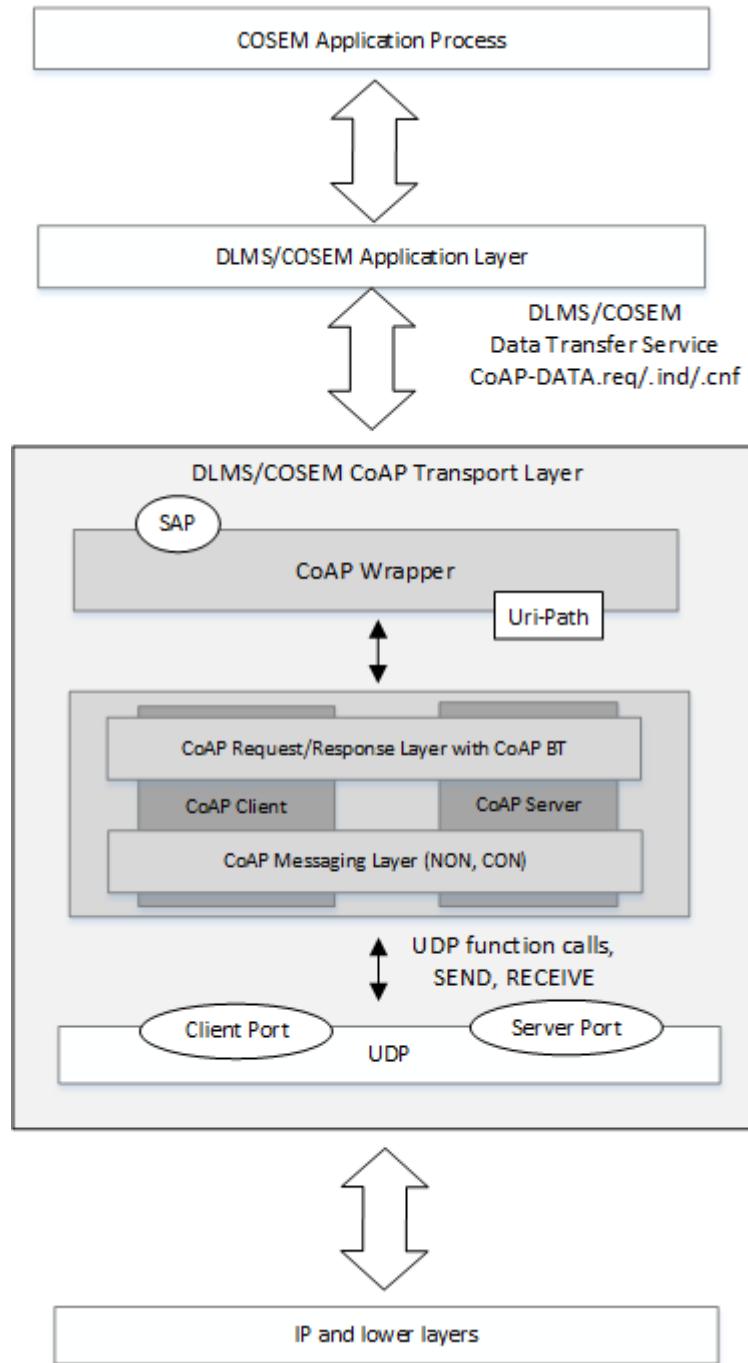


**Figure 43 – DLMS/COSEM CoAP transport protocol layer**

### 7.3.3 Structure of the DLMS/COSEM CoAP transport layer

#### 7.3.3.1 General

The structure of the DLMS/COSEM CoAP TL is shown in Figure 44.



**NOTE on Figure 44:** The Client and the Server Port may be identical. See 7.3.3.2.

**Figure 44 – Structure of DLMS/COSEM CoAP Transport Layer**

The DLMS/COSEM AL is bound to the CoAP request/response layer through the CoAP wrapper and its usage of the CoAP POST method.

The DLMS/COSEM CoAP TL provides unreliable and reliable transport services. The unreliable DLMS/COSEM transport services utilize non-confirmable (NON) CoAP messages and the reliable DLMS/COSEM CoAP TL services utilize confirmable (CON) CoAP messages with the retry mechanism provided by the CoAP messaging layer.

The CoAP wrapper layer provides the following functions:

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	105/614
-----------------------	------------	-----------------------	---------

- OSI-style data service primitives to the DLMS/COSEM AL through interaction with the CoAP request/response layer operation for CoAP POST method usage;
- DLMS/COSEM Service Access Point addressing capability, thus allowing for multiple DLMS/COSEM AEs to be hosted on the same CoAP client and server endpoints of a physical device;

NOTE For efficiency reasons, as well as for addressing simplicity, it is desirable to allow sharing of the same UDP sockets and the same CoAP client and server endpoints for multiple DLMS/COSEM AEs that reside on the same physical device.

The CoAP wrapper layer implements the stateful operation required for the provisioning of OSI-style data services on top of the stateful CoAP messaging and CoAP block transfer service in conjunction with the CoAP protocol layers.

The data field of the data service primitives of the CoAP wrapper layer carries an APDU. For further details on CoAP wrapper service primitive see 7.3.4.

For further details on the identification and addressing scheme, the CoAP Uri-Path and the UDP and IP identities, see 7.3.3.2.

For further details on the implementation of the DLMS/COSEM CoAP TL and the CoAP wrapper layer contained herein see section 7.3.5.

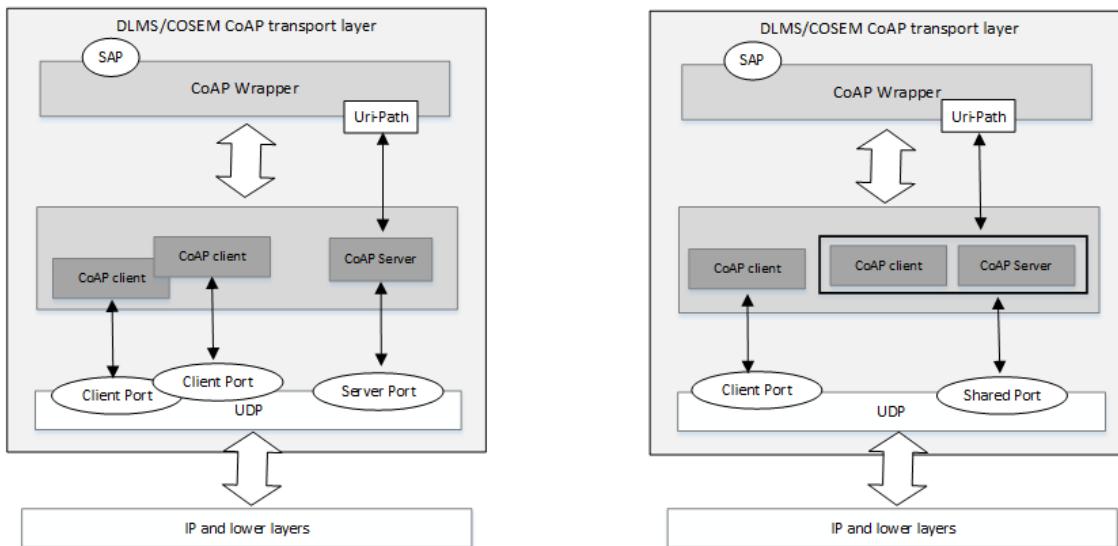
### 7.3.3.2 Identification and addressing

#### 7.3.3.2.1 CoAP endpoints within the CoAP transport layer

A CoAP client is the originating endpoint of a CoAP request and the destination endpoint of a CoAP response and implements the associated CoAP protocol operations in accordance with the CoAP protocol specification, RFC 7252. A CoAP server is the destination endpoint of a CoAP request and the originating endpoint of a CoAP response and implements the associated CoAP protocol operations. Both CoAP client and CoAP server endpoints are implemented by the DLMS/COSEM CoAP TL of a DLMS client or a DLMS server. See Figure 45.

A DLMS/COSEM CoAP TL with separate CoAP client and CoAP server endpoints with an additional Separate CoAP client endpoint

A DLMS/COSEM CoAP TL with a combined CoAP client and CoAP server endpoint with an additional separate CoAP client endpoint



NOTE on Figure 45: The usage of the Uri-Path is described in 7.3.3.2.2.

**Figure 45 – CoAP client and server endpoints within the DLMS/COSEM CoAP TL**

A DLMS/COSEM CoAP TL may implement separate CoAP client and CoAP server endpoints, i.e., with usage of different UDP ports for the client and server endpoint; see Figure 45, or a combined CoAP client-server endpoint used with a shared UDP port; see Figure 45. A combined CoAP client-server

endpoint is required for use with DLMS GBT streaming in the DLMS/COSEM CoAP communication profile.

For traffic flow differentiation and separation, a DLMS/COSEM CoAP TL may implement multiple CoAP client and CoAP server endpoints; see Figure 45.

**NOTE** Multiple client endpoints may be used in a DLMS server, for example, to implement a high priority alarm channel as well as generally to implement separate DataNotification channels from the channels used to return service invocation responses.

A CoAP server endpoint of the DLMS/COSEM CoAP transport layer is by default bound to UDP port 5683 which is reserved for CoAP services by IANA. Alternatively, it may be bound to an ephemeral UDP port.

A CoAP client endpoint of the DLMS/COSEM CoAP transport layer may be bound to an ephemeral UDP port, or, in the case of a shared CoAP client-server endpoint for DLMS/COSEM CoAP TL to port 5683.

The identity of a CoAP server endpoint of the DLMS/COSEM CoAP TL is defined by the CoAP Uri-Host and the CoAP Uri-Port. The CoAP Uri-Host may be specified as an IPv6 or IPv4 address. The CoAP Uri-Port is the server UDP port.

The DLMS/COSEM CoAP TL layer may use the same IP address for the CoAP client-server endpoints but this is not mandatory. A physical device hosting one or more DLMS servers typically uses the same IP address for each of the CoAP client-server endpoints.

The IP addresses and the ports of the CoAP server endpoints of a physical device hosting one or more DLMS servers may be retrieved from the CoAP setup objects and the associated UDP setup objects, see DLMS UA 1000-1 Part 2 Ed.15:2021,4.9.1

When there are multiple CoAP endpoints to choose from for the transmission of a DataNotification APDU, the CoAP endpoint to use is specified in the Push setup object, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.8.

Diagnostic information for CoAP protocol layer operation within the DLMS/COSEM CoAP TL of a DLMS server may be retrieved from CoAP diagnostic object, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.9.9).

### 7.3.3.2.2 DLMS/COSEM AL identification within the CoAP transport layer

The CoAP protocol layer instance and the CoAP client and server endpoints of the DLMS/COSEM CoAP TL may be shared with other applications running on the entity.

The CoAP Uri-Path within the CoAP URI identifies the target application layer resource of an incoming CoAP request. When multiple application layer resources are hosted on the same CoAP server endpoint they are identified by their individual CoAP Uri-Path segments. When only one application layer resource is hosted by a CoAP server endpoint, the Uri-Path segment may be the unspecified "empty" path. A specified Uri-Path is carried in the Uri-Path option within the CoAP messaging layer. See 7.3.5.4.2.3.

By default, a DLMS/COSEM AL, whether it be a DLMS client AL or a DLMS server AL, uses the URI-Path: "dlms".

The Uri-Path of the DLMS/COSEM AL of a physical device may be retrieved from the CoAP setup object, see DLMS UA 1000-1 Part 2 Ed.15:2021,4.9.8.

### 7.3.3.2.3 DLMS/COSEM CoAP transport layer SAPs

The reserved client SAPs and server SAPs used with the DLMS/COSEM CoAP communication profile are specified in Table 3.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	107/614
-----------------------	------------	-----------------------	---------

**Table 3 – Reserved SAP numbers in the DLMS/COSEM CoAP communication profile**

<b>Client side reserved SAPs</b>	
	<b>Client SAP</b>
Client Management Process	0x01
Public Client	0x10
Open for SAP assignment	0x02...0x0F
	0x1...0xFF
<b>Server side reserved SAPs</b>	
	<b>Server SAP</b>
Management Logical Device	0x01
Reserved	0x02...0x0F
Open for SAP assignment	0x10...0x7E
All-station	0x7F

The SAPs are carried in the DLMS/COSEM CoAP communication profile between the CoAP endpoints of the DLMS/COSEM CoAP TL in the dedicated CoAP wrapper fields of the CoAP-PDU (see 7.3.5). These are exchanged between the CoAP wrapper and the DLMS/COSEM AL through the CoAP-DATA service primitives.

Messages containing SAPs that are invalid within the DLMS/COSEM CoAP communication profile are discarded by the receiving CoAP wrapper layer which may respond by returning a CoAP error to the sending CoAP wrapper. See 7.3.5.6.

#### 7.3.4 Service specification for the DLMS/COSEM CoAP transport layer

##### 7.3.4.1 General

The connection-less DLMS/COSEM CoAP TL provides the same set of services at both the DLMS client and DLMS server side; see Figure 46.

The DLMS/COSEM CoAP TL provides data transport services to the DLMS/COSEM AL through three data services primitives:

1) CoAP-DATA.request:

Used to convey an APDU from the sender DLMS/COSEM application layer to the DLMS/COSEM CoAP TL for delivery towards the receiver DLMS/COSEM application layer.

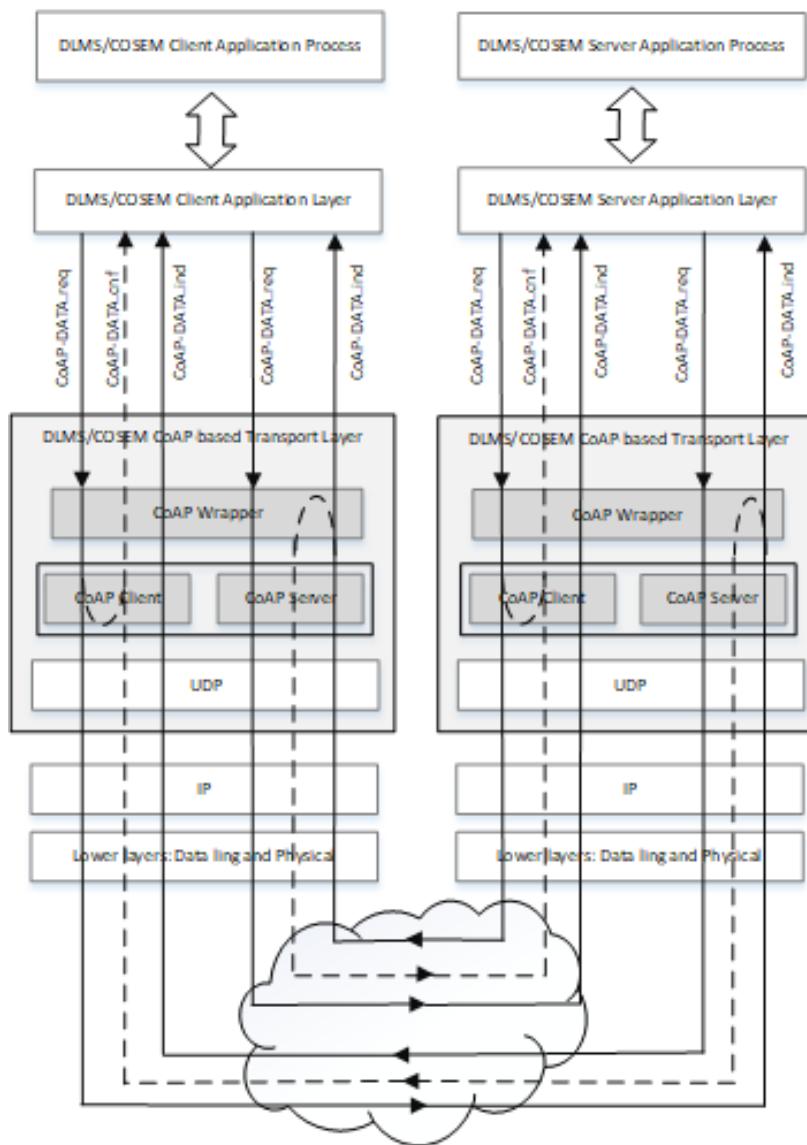
2) CoAP-DATA.indication:

Used to convey a received APDU from the DLMS/COSEM CoAP TL to the DLMS/COSEM application layer.

3) CoAP-DATA.confirm:

Used to convey DLMS/COSEM CoAP TL delivery success or failure indication back to the sender DLMS/COSEM application layer when sending an APDU was attempted.

The CoAP-DATA.confirm service primitive is optional to support the unreliable DLMS/COSEM CoAP TL transport service.



**Figure 46 – Services of the connection-less DLMS/COSEM CoAP transport layer**

Figure 46 shows how the primitives are used and implemented through the CoAP wrapper and the underlying CoAP client and CoAP server protocol entities. Figure 46 illustrates the situation where DLMS response APDUs are returned as CoAP responses within an existing CoAP request/response context, originating through the CoAP server entity. Confirmed service invocation requests are sent as CoAP requests originating from the CoAP client entity. Under certain situations (such as in the case of DLMS GBT streaming over DLMS/COSEM CoAP TL with usage of combined CoAP client and server endpoints), DLMS response APDUs may be returned through new CoAP requests originating from the combined CoAP client and server entity and DLMS request APDUs may be sent as CoAP responses originating from the combined CoAP client and server entity see 7.3.6.6.3.

The dotted remote loop-backs in Figure 46 illustrate the situation when a DLMS/COSEM CoAP TL confirmation message is autonomously generated to confirm that an APDU has been successfully received by the receiving CoAP wrapper. This confirmation message is returned to the sending CoAP client where it will be conveyed as a positive DLMS/COSEM CoAP TL confirmation to the sending DLMS/COSEM AL through the CoAP-DATA.confirm primitive.

The dotted local loop-backs in Figure 46 illustrate the situation where the reliable transfer of an APDU fails. In this case, a negative confirmation is generated by the local DLMS/COSEM CoAP TL and returned to the sending DLMS/COSEM AL through the CoAP-DATA.confirm primitive.

Positive DLMS/COSEM CoAP TL confirmations are only provided by the reliable DLMS/COSEM CoAP TL transport service. Negative DLMS/COSEM CoAP TL confirmations may be provided by the reliable and the unreliable DLMS/COSEM CoAP TL transport service.

The implementation of the primitives through the underlying CoAP wrapper, and the underlying CoAP protocol layer is described in detail in 7.3.5. The usage of the primitives by the DLMS AL and the resulting CoAP message transfer is described in 7.3.6. The general mapping of the DLMS/COSEM AL service primitives to the DLMS/COSEM CoAP TL service primitives is described in 10.4.6.

#### 7.3.4.2 The DLMS/COSEM CoAP-DATA service primitives

##### 7.3.4.2.1 CoAP-DATA.request

###### 7.3.4.2.1.1 Function

This primitive is the service request primitive for the DLMS/COSEM CoAP data transfer service.

###### 7.3.4.2.1.2 Semantics of the service primitive

The primitive shall provide parameters as follows:

```
CoAP-DATA.request ( {
    Transport_Mode,
    Local_SAP,
    Remote_SAP,
    Local_IP_address [Optional Use],
    Local_Port [Optional Use],
    Remote_IP_address,
    Remote_Port,
    Remote_Path [Optional Use],
    Response_Mode,
    Request_ID [Optional Use],
    Data_Length,
    Data
})
```

NOTE Service parameters marked with [Optional Use] are used only in certain situations as described under 7.3.4.2.1.3.

Where:

- Transport\_Mode indicates whether the APDU is to be sent as a CoAP Confirmable message for reliable CoAP transport operation (Transport-Mode set as “RELIABLE”) or as a Non-Confirmable message for unreliable CoAP transport operation (Transport-Mode set as “UNRELIABLE”);
- Local\_SAP indicates the SAP of the local DLMS/COSEM AE;
- Remote\_SAP indicates the SAP of the remote DLMS/COSEM AE;
- Local\_IP\_address and the Local\_Port indicate the CoAP endpoint of the sending DLMS/COSEM CoAP TL;
- Remote\_IP\_address and Remote\_Port indicate the CoAP endpoint for the receiving DLMS/COSEM CoAP TL;
- Remote\_Path indicates the Uri-Path of the receiving DLMS/COSEM CoAP TL peer. The Remote\_Path shall not be set and is ignored if Response\_Mode is set to “RESPONSE”;
- Response\_Mode indicates whether a DLMS/COSEM response APDU is expected to be returned. It takes the values: “CONFIRMED”, “UNCONFIRMED”, “RESPONSE”. For further specification see 7.3.4.2.1.3;
- Request\_ID identifies the particular data request operation. The Request\_ID will be returned in a potential resulting CoAP-DATA.confirm primitive indicating the success or the failure of the DLMS/COSEM CoAP TL to transfer the APDU given in the Data parameter. The Request\_ID is

specified to support return of DLMS/COSEM CoAP TL confirmation on a per APDU basis in situations where multiple APDUs carrying requests have been sent and the DLMS/COSEM CoAP TL confirmations are outstanding. The following applies:

- In case the Request\_ID is left unspecified, the Request\_ID is left unspecified in a potential resulting CoAP-DATA.confirm primitive generated by the DLMS/COSEM CoAP TL implementation;
- The Request\_ID may be left unspecified if the Transport\_Mode is set to UNRELIABLE and the DLMS/COSEM CoAP TL implementation does not support the CoAP-DATA.confirm primitive for this mode of operation;
- In case the CoAP-DATA.confirm primitive is not supported by the DLMS/COSEM CoAP TL service, a specified Request\_ID identifier is ignored by the CoAP wrapper.
- Data\_Length indicates the length of the Data parameter in bytes;
- Data contains the APDU to be transferred to the receiving DLMS/COSEM AL.

#### 7.3.4.2.1.3 Use

The CoAP-DATA.request primitive is invoked by the sending DLMS/COSEM AL in the following manner in the following use case situations:

- 1) When sending a DLMS/COSEM service invocation request APDU to a single receiving DLMS/COSEM AE, or, in the case of multicasting or broadcasting, to multiple receiving DLMS/COSEM AEs. The following applies for this use case:
  - a) The Remote\_Path is specified as the Uri-Path of the receiving DLMS/COSEM CoAP TL peer;
  - b) The Local\_Port and Local\_IP\_address are optional to specify. If not specified they will be chosen by the underlying UDP/IP layer. In case the data request operation needs to be carried out in the context of an existing Application Association using a particular set of Local\_Ports and Local\_IP\_addresses, these must be specified;
  - c) The Response\_Mode:
    - i. Generally, the Response\_Mode shall be set to CONFIRMED when sending an APDU carrying a confirmed service invocation request;
    - ii. Generally, the Response\_Mode shall be set to UNCONFIRMED when sending an APDU carrying an unconfirmed service invocation request;
    - iii. To support optimal CoAP messaging layer operation in situations with GBT streaming this shall be further qualified as follows:
      - The Response\_Mode may be set to CONFIRMED by a requesting DLMS AL when sending a GBT APDU for which a response is expected from a GBT streaming procedure perspective;
      - The Response\_Mode may be set to UNCONFIRMED by a requesting DLMS AL when sending a GBT APDU for which no response is expected from a GBT streaming procedure perspective.
- 2) When sending a DLMS/COSEM service invocation response APDU to a single receiving DLMS/COSEM AE. The following applies for this use case:
  - a) The Remote\_Path is not specified;
  - b) The Local\_Port and Local\_IP\_address shall be specified and shall match the corresponding parameters received in the CoAP-DATA.indication primitive with the confirmed APDU to which this is a response;
  - c) The Transport\_mode, Local\_SAP, Remote\_SAP, Remote\_IP\_address, Remote\_Port shall match the corresponding parameters received in the CoAP-DATA.indication primitive with the confirmed APDU to which this is a response;
  - d) The Response\_Mode:

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	111/614
-----------------------	------------	-----------------------	---------

- i. Generally, the Response\_Mode shall be set to RESPONSE when sending a service invocation response APDU;
  - The Remote\_Path shall not be specified in this case, but will be chosen implicitly or explicitly by the layers below, see 7.3.5.7.5;
- ii. To support optimal CoAP messaging layer operation in situations with GBT streaming this shall be further qualified as follows:
  - The Response\_Mode may be set to CONFIRMED by a responding DLMS AL when sending a GBT APDU for which a response is expected from a GBT streaming procedure perspective;
    - The Remote\_Path is specified as the Uri-Path of the receiving DLMS/COSEM CoAP TL peer.

The implementation of and the usage of the CoAP-DATA.request service primitive is further described in 7.3.5 and in 7.3.6.

#### **7.3.4.2.2 CoAP-DATA.indication**

##### **7.3.4.2.2.1 Function**

This primitive is the service indication primitive for the DLMS/COSEM CoAP data transfer service.

##### **7.3.4.2.2.2 Semantics of the service primitive**

The primitive shall provide parameters as follows:

```
CoAP-DATA.indication ( 
  Transport_Mode,
  Local_SAP,
  Remote_SAP,
  Local_IP_address,
  Local_Port,
  Remote_IP_address,
  Remote_Port,
  Data_Length,
  Data
)
```

Where:

- Transport\_Mode indicates whether the APDU is received as a CoAP Confirmable message for reliable CoAP transport operation (Transport-Mode set as “RELIABLE”) or as a Non-Confirmable message for unreliable CoAP transport operation (Transport-Mode set as “UNRELIABLE”);
- Local\_SAP indicates the SAP of the local DLMS/COSEM AE;
- Remote\_SAP indicates the SAP of the remote DLMS/COSEM AE;
- Local\_IP\_address and Local\_Port provide the details of the receiving CoAP endpoint. Remote\_IP\_address and Remote\_Port provide the details of the sending CoAP endpoint. Data\_Length indicates the length of the data parameter in bytes;
- Data contains the APDU received from the sending DLMS/COSEM AL;

##### **7.3.4.2.2.3 Use**

The CoAP-DATA.indication primitive is generated by the CoAP wrapper layer to indicate to the receiving DLMS/COSEM AE that an APDU from a sending DLMS/COSEM AE has been received. The primitive is generated following the receipt of a complete APDU (possibly received through CoAP BT) by the DLMS/COSEM CoAP TL provided that both the Local\_SAP and the Remote\_SAP contain valid SAPs (see Table 3). The implementation of and the usage of the CoAP-DATA.indication service primitive is further described in 7.3.5 and in 7.3.6.

**7.3.4.2.3 CoAP-DATA.confirm****7.3.4.2.3.1 Function**

This primitive is the service confirm primitive for the DLMS/COSEM CoAP data transfer service.

**7.3.4.2.3.2 Semantics of the service primitive**

The primitive shall provide the following parameters:

**CoAP-DATA.confirm (**

Local_SAP,	[Optional Use],
Remote_SAP,	[Optional Use],
Local_IP_address	[Optional Use],
Local_Port,	[Optional Use],
Remote_IP_address,	
Remote_Port,	
Request_ID	[Optional Use],
Result	

)

Service parameters marked with [Optional Use] are optional to specify in certain situations. Their usage is further detailed in 5.7.3.3.2.2.

Where:

- Local\_SAP indicates the SAP of the local DLMS/COSEM AE;
- Remote\_SAP indicates the SAP of the remote DLMS/COSEM AE;
- Local\_IP\_address and Local\_Port indicate the CoAP endpoint of the local DLMS/COSEM CoAP TL;
- Remote\_IP\_address and Remote\_Port indicate the CoAP endpoint for the intended receiving DLMS/COSEM CoAP TL;
- Request\_ID carries the identifier that the DLMS/COSEM AL provided in the CoAP-DATA.request primitive that carried the APDU that this transfer layer confirmation primitive relates to. If no Request\_ID identifier was specified in the CoAP-DATA.request primitive, the Request\_ID is left unspecified;
- Result indicates by "REMOTE OK" that a DLMS/COSEM CoAP TL confirmation has been received from the remote protocol layer endpoint. The Result parameter indicates by "NOT OK" if delivery of the APDU to the remote side has failed.

**7.3.4.2.3.3 Use**

The CoAP-DATA.confirm primitive is generated by the CoAP wrapper layer to positively ("REMOTE OK") or negatively ("NOT OK") confirm to the service user DLMS/COSEM AL the transfer of an APDU provided in a previous CoAP-DATA.request.

For the unreliable DLMS/COSEM CoAP TL transport service, only failure indications can be generated.

The support of the primitive for the unreliable DLMS/COSEM CoAP TL transport service is optional.

The DLMS/COSEM CoAP TL mechanisms supporting the notification of positive ("REMOTE OK") or negative ("NOT OK") confirmations are described in further detail in 7.3.5 and in 7.3.6.

**7.3.5 Protocol specification of the DLMS/COSEM CoAP transport layer****7.3.5.1 General**

The DLMS/COSEM CoAP TL provides OSI type services to the service user DLMS/COSEM AL. It comprises the following:

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	113/614
-----------------------	------------	-----------------------	---------

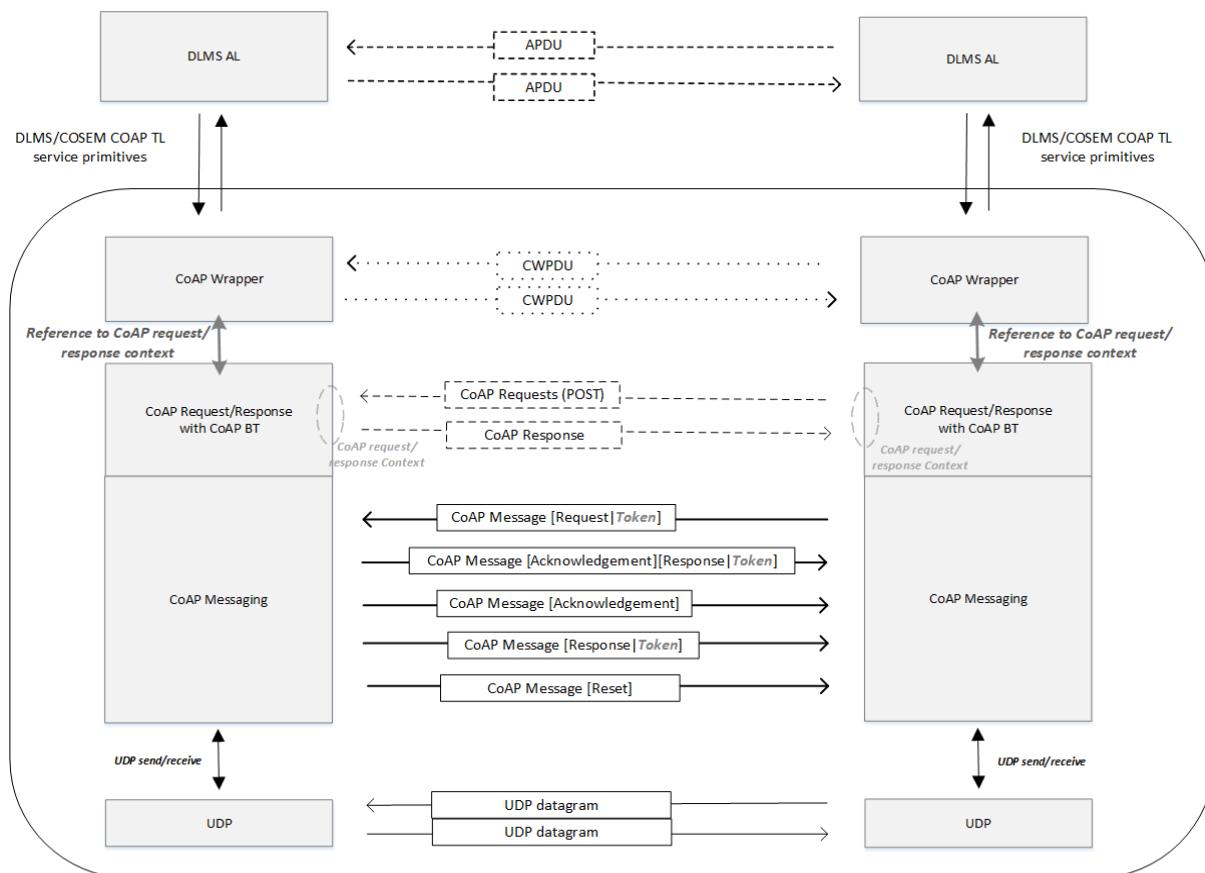
- The DLMS/COSEM CoAP wrapper layer that transforms the OSI style CoAP-DATA services to CoAP request/response function calls. It is a stateful entity that mediates between the stateful CoAP request/response layer and OSI service layer. The CoAP wrapper implements the DLMS/COSEM CoAP TL confirmations of the reliable DLMS/COSEM CoAP TL;
- The CoAP protocol layer that uses a selected subset of the features specified in RFC 7252. For block-wise transfers it uses a subset of the features specified in RFC 7959;
- The UDP layer as specified in RFC 768.

Only the DLMS/COSEM COAP wrapper layer is DLMS/COSEM specific. Therefore, for implementing the CoAP layer and the UDP layer within the DLMS/COSEM CoAP TL protocol stack, standard CoAP and UDP protocol layer implementations may be used.

### 7.3.5.2 The DLMS/COSEM CoAP TL Protocol Data Unit (CoAP-PDU)

This clause describes the DLMS/COSEM CoAP TL Protocol Data Unit (CoAP-PDU) and the coupling of the DLMS CoAP wrapper layer to the CoAP request/response layer.

The DLMS/COSEM CoAP TL protocol stack is shown in Figure 47.



**Figure 47 – The DLMS/COSEM CoAP TL Protocol Stack**

The DLMS/COSEM CoAP TL PDU is a UDP datagram that carries a CoAP message as its payload. The CoAP message carries the CoAP header and the DLMS/COSEM CoAP Wrapper PDU (CWPDU). The CWPDU carries the DLMS/COSEM APDU as its payload, as well as DLMS/COSEM CoAP TL control information. See 7.3.5.3.

A CWPDU is carried as a CoAP request (POST method) or a CoAP response in a CoAP message. The CWPDU may be empty in which case it is carried in a CoAP message as an empty response payload. See 7.3.5.6

In addition to CoAP messages carrying CWPDUs, exchanged in between the DLMS/COSEM CoAP TL peers, a number of other CoAP messages:

- CoAP messages carrying CoAP error responses;
- CoAP acknowledgement messages (in the case of reliable transport service operation);
- CoAP reset messages, are exchanged to support the CoAP protocol layer operation.

For efficient and reliable transport service operation, CoAP responses, in form of CWPDUs or CoAP error responses, are carried piggybacked within CoAP acknowledgement messages when possible. See 7.3.5.4.3 and 7.3.6. A CWPDU is transferred between the CoAP endpoints within a CoAP request/response context of the CoAP request/response protocol layer. The CoAP Token identifies the CoAP request/response context at the CoAP protocol level.

CWPDUs interchanged in between the CoAP wrapper and the CoAP request/response layer are logically interchanged in the context of the corresponding CoAP request/response context in the CoAP request/response layer. The exact realization of the “CoAP request/response context reference” at the application programming interface, provided by a CoAP protocol layer, is an implementation decision.

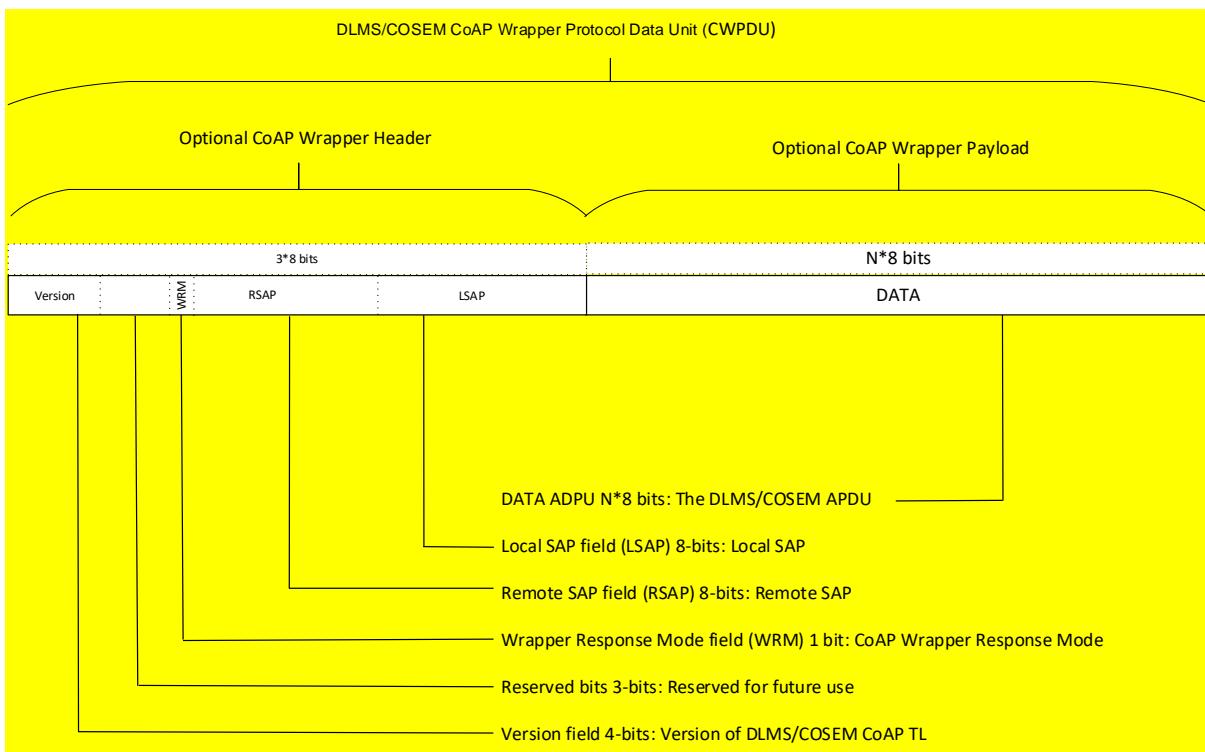
The CoAP wrapper maintains CoAP wrapper request/response context state for each of the (open) CoAP request/response contexts, both those initiated by itself towards a remote endpoint for the transfer of a CWPDU towards the remote peer and for each (open) CoAP request/response context initiated towards itself from a remote endpoint for the transfer of a CWPDU towards itself from the peer. The CoAP wrapper request/response context and state machine are described in 7.3.5.7.

### **7.3.5.3 The DLMS/COSEM CoAP Wrapper Protocol Data Unit (CWPDU)**

The DLMS/COSEM CoAP Wrapper Protocol Data Unit (CWPDU) is composed of an optional DLMS/COSEM CoAP wrapper header and its payload, the APDU. See Figure 48.

The DLMS/COSEM CoAP wrapper header is used by the CoAP wrapper only when an APDU is carried in a CoAP request. When the APDU is carried in a CoAP response no DLMS/COSEM CoAP wrapper header is included within the CWPDU since the relevant CoAP wrapper header contents (local and remote SAP) can be derived at the receiving CoAP wrapper layer from the state of the CoAP wrapper request/response context. See 7.3.5.7.

As a special case, an empty CWPDU is used by the CoAP wrapper to implement the DLMS/COSEM CoAP TL confirmations of the reliable DLMS/COSEM CoAP TL. See 7.3.5.6.



**Figure 48 – The DLMS/COSEM CoAP Wrapper Protocol Data Unit (CWPDU).**

The DLMS/COSEM CoAP wrapper header fields carry the following parameters:

- The DLMS/COSEM CoAP TL version: The DLMS/COSEM CoAP TL version implemented by the sending endpoint formatted as non-negative integer (unsigned) between 0 and 15. The DLMS/COSEM CoAP TL version specified here is version 0. The DLMS/COSEM CoAP TL version supported by a DLMS server may be retrieved from the “CoAP setup” object, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.9.8;
- Reserved bits: Bits reserved for future use;
- The CoAP Wrapper Response Mode (WRM) used to notify the receiving CoAP wrapper layer whether a DLMS AL or DLMS AP response is expected to be generated by the receiving DLMS/COSEM AE. (0) means that a DLMS response is expected to be returned by the receiving DLMS AE. (1) means that no DLMS response is expected to be generated by the receiving DLMS/COSEM AE. The CoAP Wrapper Response Mode is used for the implementation of the DLMS/COSEM CoAP TL confirmations of the reliable DLMS/COSEM CoAP TL. Its usage is specified in 7.3.5.6;
- The Remote SAP identifying the Application Process of the receiving endpoint;
- The Local SAP identifying the Application Process of the sending endpoint.

#### 7.3.5.4 The Constrained Application Protocol (CoAP)

##### 7.3.5.4.1 General

The constrained application protocol operation within the DLMS/COSEM CoAP TL is based on the standard CoAP protocol layer operation of the CoAP messaging layer and the CoAP request/response and block transfer layers as specified by RFC 7252 and RFC 7959.

The CoAP wrapper operates as the CoAP application layer instance.

This subclause 7.3.5.4 describes and specifies aspects of the CoAP protocol layer operation that are relevant to its use within the DLMS/COSEM CoAP TL as determined by the operation of the CoAP wrapper.

For a comprehensive description of the CoAP protocol see RFC 7252 and RFC 7959.

Error handling is described separately in 7.3.5.6.

For further details on the resulting DLMS/COSEM CoAP transport layer data transfer operation see 7.3.6.

#### 7.3.5.4.2 The CoAP Message

A CoAP message is encoded in simple binary format. The message is composed of a fixed-size 4-byte header, a variable-length Token value of 0-8 bytes, zero or more TLV-encoded options optionally followed by the payload.

A non-empty CWPDU is carried as payload in the CoAP message.

##### 7.3.5.4.2.1 CoAP Message Header Fields

The Type and the Code field of the fixed-size 4-byte CoAP message header together indicate the CoAP message type.

For other fields of the CoAP message header and their usage see RFC 7252.

###### 7.3.5.4.2.1.1 CoAP Header Type field

The CoAP header Type field (2 bit) values specified by RFC 7252 are:

- Type = CON (0) for confirmable messages carrying request or responses;
- Type = NON (1) for non-confirmable messages carrying request or responses;
- Type = ACK (2) for CoAP acknowledgement messages (may embed piggybacked responses);
- Type = RST (3) for CoAP reset messages.

###### 7.3.5.4.2.1.2 CoAP Header Type field use

The CoAP wrapper does not directly manipulate the CoAP header type field, it is set by the CoAP protocol implementation depending on the type of message issued.

For any given CWPDU or error response to be sent in a new originating CoAP request/response context, the CoAP wrapper layer shall specify whether the CoAP request/response context shall use the reliable or the non-reliable CoAP messaging layer.

A CWPDU or an error response exchanged by the reliable DLMS/COSEM CoAP TL shall use the reliable CoAP messaging layer. The CWPDU or the error response is carried in a CoAP message of type CON or ACK. Whether carried in type CON or ACK is determined by the CoAP messaging layer, see 7.3.5.4.3.

A CWPDU exchanged by the unreliable DLMS/COSEM CoAP TL shall use the unreliable CoAP messaging layer and is carried in a CoAP message of type NON by the CoAP messaging layer.

For the usage of reset messages in the CoAP messaging layer see RFC 7252.

###### 7.3.5.4.2.1.3 CoAP header code field

The CoAP header Code field (8 bit) values are divided in the following classes:

- Class (0): Carries request method codes;
- Class (2): Carries success response codes;
- Class (4): Carries CoAP client error response codes;
- Class (5): Carries CoAP server error response codes.

The code field is formatted as a 3 bit class field (c.) followed by a 5 bit code detail field (dd) specifying the sub-codes within each class.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	117/614
-----------------------	------------	-----------------------	---------

#### 7.3.5.4.2.1.4 CoAP Request method codes

The request method codes used in CoAP messages of the DLMS/COSEM CoAP TL are the following:

**Table 4 – CoAP Request method codes**

Request method	Meaning	Use
0.02	POST method	Specified by the CoAP wrapper with a CWPDU to be transferred in a new CoAP request/response context
0.00	ACK message without piggybacked response	Set by the reliable CoAP messaging layer when sending ACKs with no piggybacked response

#### 7.3.5.4.2.1.5 CoAP Success Response codes

The success response codes in use by the CoAP wrapper are the following:

**Table 5 – CoAP Success Response codes**

Success Response code	Meaning	Use
2.04	Success (Changed)	Specified by the CoAP wrapper with a CWPDU to be transferred as a response in an existing request/response context

#### 7.3.5.4.2.1.6 CoAP Client and server error Response codes

The client error and server error response codes are populated either by the CoAP protocol layers layer or by the CoAP wrapper depending on the error condition. See 7.3.5.5.

#### 7.3.5.4.2.2 Token

The Token is used to match a response with a request (RFC 7252). The Token identifies the CoAP request/response context. The Token is set in a request and reflected in the matching response.

A CoAP BT implementation may choose to use the same or different Tokens for the transfer of the individual blocks of the CWPDU each of which are carried as CoAP request or CoAP response CoAP message payloads. For further details on the CoAP BT operation of the DLMS/COSEM CoAP TL see 7.3.5.4.4.

#### 7.3.5.4.2.2.1 Token Length

The Token length used by the CoAP request/response layer of a DLMS/COSEM CoAP TL implementation is recommended to be restricted to 0-4 bytes to balance the Token transmission costs up against the risk of context mismatching or duplicate detection failure that can occur when a Token is reused in between the same CoAP endpoints. For further reference see RFC 7252.

The Token length to use by the CoAP protocol layer of the DLMS/COSEM CoAP TL of a DLMS server may be specified in the CoAP setup object, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.9.8.

### 7.3.5.4.2.3 Options

The following CoAP options are used by the DLMS/COSEM CoAP TL protocol operation; See Table 6.

**Table 6 – CoAP Options used by the DLMS/COSEM CoAP TL**

Option	Value	Use	Reference
Uri-Path	"dlms" by default	Specified by the CoAP wrapper with a CWPDU to be transferred in a new CoAP request/response context if non-empty Uri-Path	RFC 7252
Content-Format	application/octet-stream – ID 42	Specified by the CoAP wrapper with a non-empty CWPDU to be transferred in either a new CoAP request/response context or as a response in an existing CoAP request/response context.	RFC 7252
Block1	Dynamic	Specified by CoAP BT	RFC 7959
Block2	Dynamic	Specified by CoAP BT	RFC 7959

The Uri-Path value is used to communicate the Uri-Path of the receiving DLMS/COSEM CoAP TL peer, see 7.3.3.2.2. The value is specified by the CoAP wrapper for a CWPDU handed over to the CoAP request/response layer for transfer within a new CoAP request/response context. The CoAP protocol layer embeds a non-empty Uri-Path value in a Uri-Path option within the CoAP message. The empty path is implied as default if no Uri-Path option is carried with the CoAP request.

The specified Content-Format option value is set by the CoAP wrapper for every non-empty CWPDU handed over to the CoAP request/response layer for transmission. The Accept option, communicating the content-format accepted by a CoAP client, see RFC 7252, section 5.10.4, need not be used as it can be assumed that DLMS peers accept xDLMS APDU octet-stream format.

The use of the Block1 and Block2 options are controlled by the CoAP BT layer within the CoAP request/response layer. For the usage of these options see the 7.3.6.5 and RFC 7959.

This specification does not specify the CoAP protocol layer or the CoAP wrapper layer of the DLMS/COSEM CoAP TL layer to use other CoAP options than those listed in Table 6. However, it is allowed<sup>1</sup> for the CoAP protocol layer or the CoAP wrapper layer of the DLMS/COSEM CoAP TL layer to use more options than those specified here. The CoAP wrapper layer of the DLMS/COSEM CoAP TL implementation shall be prepared to receive options other than those listed in Table 6 and shall behave in accordance with RFC 7252 for handling of unrecognized options when doing so.

NOTE 1: A CoAP wrapper may, for example, exploit the RFC 7252 Size 1 option and the RFC 7252 Max-Age option for enhanced error notification; The Size 1 option, for example, could be used to communicate the receiver\_max\_pdu\_size of the receiving DLMS AL in situations where the received APDU is rejected with "Request Entity Too Large" error due to it exceeding this size. See 7.3.5.5 . The Max-Age option, for example, could be used to guide DLMS AL retransmission in situations where acceptance of the incoming APDU is rejected with "5.03 Service Unavailable" error due to temporary CoAP wrapper or DLMS AL unavailability (overload). See 7.3.5.5. Any usage of these options is outside the scope of this specification.

### 7.3.5.4.3 CoAP retransmission and response piggybacking

#### 7.3.5.4.3.1 General

When a CWPDU is transferred within a new CoAP request/response context supported by the reliable CoAP messaging layer (i.e., as through confirmable (CON) CoAP messages) then, as specified by RFC 7252, the CoAP messaging layer will continue to retransmit the CoAP request message until it is acknowledged by the CoAP server endpoint. This may be either in form of the return of a separate CoAP acknowledgement message or by return of a CWPDU or an error response piggybacked on a CoAP acknowledgement message.

The use of piggybacked response payloads (or errors) over reliable CoAP transport allows resources to be saved in the network both at the requesting CoAP client and at the responding CoAP server endpoints.

Piggybacking of a CWPDU response payload (or error response) is assumed to be supported by the CoAP messaging layer of the DLMS/COSEM CoAP TL.

The CoAP wrapper shall return an error response, or a response CWPDU as soon as it is available from DLMS AL to the reliable CoAP messaging layer. The reliable CoAP messaging layer of the CoAP server endpoint of the CoAP messaging layer will wait for the possibility to piggyback a response payload (or an error response) with a CoAP acknowledgment message. However, if the opportunity to piggyback a response payload (or an error response) with the acknowledgement does not occur within a timeout period after the arrival of the request, the reliable messaging layer of the CoAP server endpoint will return a CoAP acknowledgement message without any piggybacked response.

If a response payload or an error response is provided after the timeout period has expired, it will be returned in a separate CoAP response message.

The timeout period is specified in RFC 7252 to be identical to the CoAP ack\_timeout parameter. The ack\_timeout defines the lower limit of the initial retransmission delay of the CoAP messaging layer. (See also Table 5.) It is common however for CoAP protocol implementations to support a separate configuration of this timeout period set to a value substantially lower than the ack\_timeout parameter value in order to prevent the occurrence of spurious retransmissions. For this purpose, separate configuration of this timeout period shall be provided, when supported, through the delay\_ack\_timeout parameter of the DLMS/COSEM CoAP setup interface class, see 5.7.4.4.3.2. and DLMS UA 1000-1 Part 2 Ed.15:2021, 4.9.8.

The reliable CoAP messaging layer is defined in RFC 7252. This specifies that a client will continue to retransmit an unacknowledged CoAP message with exponentially increasing delays until the max\_retransmit limit is reached after which the retries are abandoned.

A CoAP response that is returned piggybacked on a CoAP acknowledgement message automatically inherits the reliability associated with the reliability of the return CoAP acknowledgement message itself as polled for by the messaging layer of the CoAP client.

A separately returned response will be transmitted by the reliable CoAP messaging layer of the CoAP server until a CoAP acknowledgment message is returned from the CoAP client messaging layer. If the number of transmissions exceeds the value of the max\_retransmit parameter of the reliable CoAP messaging layer of the CoAP server endpoint the CoAP server will abandon the transmission of the separate CoAP response.

Message sequence flows depicting the retransmission operation of the reliable CoAP messaging layer and the piggybacked or separate return of responses within the DLMS/COSEM CoAP TL are given in 7.3.6.3.

#### 7.3.5.4.3.2 CoAP Retransmission Parameters

The reliable CoAP messaging layer within the DLMS/COSEM CoAP TL operates with a number of parameters controlling the CoAP retransmission algorithm as defined by RFC 7252.

The retransmission parameters exposed for modelling through the DLMS/COSEM CoAP setup interface class (see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.9.8.) are listed in Table 7.

120/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

For a comprehensive description of the parameters, see RFC 7252.

**Table 7 – CoAP retransmission parameters**

Parameter	Description	Controlled by	Reference
<code>ack_timeout</code>	<p>The minimum initial ACK timeout for confirmable messages.</p> <p>The initial_ack_timeout is a randomly chosen value in between <code>ack_timeout</code> and <code>ack_timeout x ack_random_factor</code>.</p> <p>The initial_ack_timeout is the initial retransmission delay of the reliable CoAP messaging layer.</p>	CoAP messaging layer	RFC 7552
<code>ack_random_factor</code>	The random factor to apply for randomness of the initial ACK timeout.	CoAP messaging layer	RFC 7252
<code>max_retransmit</code>	The maximum number of retransmissions for a confirmable message.	CoAP messaging layer	RFC 7252
<code>delay_ack_timeout</code>	<p>The time, in milli seconds, the CoAP messaging layer waits for the application layer to return a response before it will return an acknowledgement to prevent spurious retransmissions from peer.</p>	<p>CoAP messaging layer</p> <p>Support for this parameter is optional</p>	<p>7.3.5.4.3.1</p> <p>By RFC 7252 this parameter is not distinguished from <code>ack_timeout</code>.</p>

#### 7.3.5.4.3.3 CoAP Congestion Control Parameters

The following parameters are used in the congestion control of the RFC 7252 CoAP messaging layer, see Table 8.

**Table 8 – CoAP congestion control parameters**

Parameter	Description	Controlled by	Reference
<code>NSTART</code>	<p>The number of simultaneous outstanding CoAP request messages of either of the following form:</p> <ul style="list-style-type: none"> <li>- a CON CoAP message for which no CoAP acknowledgement has been received;</li> <li>- a NON CoAP message for which no CoAP response message has been received.</li> </ul>	CoAP messaging layer	RFC 7252
<code>PROBING_RATE</code>	Defines the average data rate in number of bytes/seconds that an endpoint shall not exceed in sending to another endpoint that does not respond.	CoAP messaging layer	RFC 7252

The parameters are exposed for configuration through the DLMS/COSEM CoAP setup interface class, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.9.8.

For a comprehensive description of the parameters see RFC 7252.

#### **7.3.5.4.4 Additional CoAP protocol parameters**

RFC 7252 defines additional protocol parameters than the ones listed in 7.3.5.4.3.2 and in 7.3.5.4.3.3. This specification assumes that RFC 7252 protocol default or vendor specific values for these parameters are used. The control of these parameter values is outside the scope of this specification.

#### **7.3.5.4.5 CoAP Block Transfer**

The CoAP block transfer operation provided within the DLMS/COSEM CoAP TL shall serve to transfer CWPDUs carrying APDUs of size larger than the MTU size but still smaller than the receiver\_max\_pdu\_size<sup>1</sup>.

NOTE 1: An APDU that is larger than the receiver\_max\_pdu\_size should not be provided to the CoAP wrapper for transmission towards a receiving DLMS AL. If it does, the APDU should be rejected either by the receiving CoAP wrapper layer (see 7.3.5.5.3.2) or by the receiving DLMS AL.

The CoAP block transfer operation provided within the DLMS/COSEM CoAP TL shall operate atomically in accordance with the semantics of the CoAP-DATA,ind primitive of the CoAP wrapper layer that requires it to handle complete APDUs. This is to ensure compliance with the semantics of the DLMS/COSEM application protocol in which a partially received APDU (i.e., the part of an APDU received through one or more CoAP blocks carrying parts of the CWPDU in which the APDU is embedded) cannot be responded to by the DLMS AL. This means that the full APDU transferred via CoAP block transfer from the sending DLMS/COSEM CoAP TL protocol layers shall be collected by the receiving CoAP wrapper layer and delivered to the receiving DLMS AL in the CoAP-DATA,ind primitive by the CoAP wrapper layer.

The CoAP block transfer layer within the DLMS/COSEM CoAP TL shall complete CoAP block transfer without undue delay as recommended in RFC 7959. Specifically, the CoAP block transfer layer within the DLMS/COSEM CoAP TL shall as a minimum clear the block transfer state and abandon the operation if no progress is observed for EXCHANGE\_LIFETIME time. For the definition of the EXCHANGE\_LIFETIME time, see RFC 7252.

Message sequence flows depicting the CoAP Block transfer operation within the DLMS/COSEM CoAP TL are given in 7.3.6.5.

#### **7.3.5 Error Handling**

##### **7.3.5.5.1 General**

Error handling is undertaken in parts by the CoAP protocol layers implemented by the CoAP messaging layer, the CoAP request/response and block transfer layers and in part by the CoAP wrapper in its capacity of operating as a CoAP application layer instance using the therefore provided mechanisms within the Constrained Application Protocol as defined by RFC 7252 and RFC 7959.

##### **7.3.5.5.2 CoAP protocol layers**

CoAP messaging layer or request/response layer errors are conveyed to the sending CoAP entity either through reset messages or through CoAP client and server error responses generated autonomously by the CoAP protocol layer entities in accordance with RFC 7252 and RFC 7959.

NOTE RFC 7252 specifies reliable return of error notification for the reliable CoAP transfer service. RFC 7252 specifies limited unreliable return of error notification for the unreliable CoAP transfer service.

##### **7.3.5.5.3 CoAP wrapper layer**

###### **7.3.5.5.3.1 Unreliable CoAP transport**

When supporting the multicast unreliable DLMS/COSEM CoAP TL service, the CoAP wrapper silently drops any CWPDU received that it is unable to process or deliver to the received DLMS AL.

### 7.3.5.5.3.2 Reliable CoAP transport

When supporting the reliable DLMS/COSEM CoAP TL service, the CoAP wrapper may return appropriate error responses in case it is unable to process an incoming CWPDU carried in a CoAP request. Such error response may aid diagnostics and may also aid proactive correction measures. Generally, CoAP client errors shall be returned when an incoming request cannot be served due to bad syntax, while CoAP server errors are returned when the CoAP wrapper fails to process an apparently valid request. The informative guidelines listed in Table 9 are provided for this purpose.

**Table 9 – CoAP wrapper error response return. [Informative]**

Error case	CoAP error Response
DLMS/COSEM CoAP TL version not supported	4.00 Bad Request
Invalid SAP	4.00 Bad Request
Transport_Mode not supported by receiving CoAP wrapper endpoint	4.00 Bad Request
Unable to serve the incoming request as the APDU therein exceeds the receiver_max_pdu_size	4.13 Request Entity Too Large
Unable to serve the incoming request due to internal processing issues in CoAP wrapper layer	5.00 Internal server error
Unable to serve the incoming request due to DLMS AL unavailability	5.03 Service Unavailable

Errors are not (cannot be) returned by the receiving CoAP wrapper to the sending CoAP wrapper peer for CWPDUs received in a CoAP response which the CoAP wrapper is unable to process and/or deliver to the DLMS AL.

### 7.3.5.5.4 Propagation of errors through CoAP wrapper layer

Error responses returned back to the sending CoAP protocol layer shall propagate to the sending CoAP wrapper in the form of resulting CoAP messaging layer delivery failures or in the direct form of the returned errors themselves whether they were generated by the receiving CoAP protocol layer or by the receiving CoAP wrapper layer.

Additionally, CoAP messaging layer delivery failures may result from the receipt of reset messages, from the abandonment of a CoAP block transfer operation (see 7.3.5.4.5), from the abandonment of reliable transmission of a CoAP message by the reliable CoAP messaging layer (see 7.3.5.4.3), and possibly as a result of local errors at the CoAP layer or errors at the underlying UDP or IP layer<sup>1</sup>.

NOTE 1 As the reliable CoAP messaging layer of the reliable DLMS/COSEM CoAP TL in any case will detect delivery failures resulting from errors in the UDP or the IP layer and as failure notification for the unreliable DLMS/COSEM CoAP TL is optional there is no requirement for UDP or IP layer failures to propagate to the CoAP layer or the CoAP wrapper layer.

CoAP protocol layer delivery failures of all CWPDUs transmitted in a locally originating CoAP request/response context through the reliable CoAP messaging layer shall propagate from the CoAP protocol layer to the CoAP wrapper layer so that it may propagate the error notification to the DLMS AL as appropriate, see 7.3.5.6.1.1.

NOTE 2 The exact manner in which delivery failures propagate to the CoAP wrapper is an implementation decision depending on the application programming interface provided by the underlying CoAP protocol layer.

## 7.3.5.6 DLMS/COSEM CoAP TL confirmations

### 7.3.5.6.1 General

The CoAP wrapper request/response context (see 7.3.5.7) maintains the state of a given Request\_ID for any outstanding CWPDU transmitted in a locally originating CoAP request/response context for

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	123/614
-----------------------	------------	-----------------------	---------

which it has not received a CoAP response so that it may return the Request\_ID with the CoAP-DATA.confirm primitive when returning a negative or a positive DLMS/COSEM CoAP TL confirmation.

It is optional for the CoAP wrapper to support return of DLMS/COSEM CoAP TL confirmations for the unreliable DLMS/COSEM CoAP TL and thus it is optional for the CoAP wrapper to maintain state of the Request\_ID for outstanding CWPDU transmitted unreliably. Only negative transport layer confirmation can be provided in form of failure indications for the unreliable DLMS/COSEM CoAP TL.

#### 7.3.5.6.1.1 CoAP transport layer failure indications

A failure indication may be in the form of a returned error response or in form of a CoAP messaging layer delivery failure, see 7.3.5.5.4

If a delivery failure indication is received from the CoAP protocol layer for a CWPDU transmitted in a locally originating CoAP request/response context, the CoAP wrapper communicates the failure to deliver the associated APDU, through the CoAP-DATA.confirm primitive with Result "NOT OK" and with a Request\_ID matching the Request\_ID provided with the APDU in the CoAP-DATA.request primitive. See 7.3.5.7.5.

This functionality is optional to support for a CWPDU transmitted in a locally originating CoAP request/response context through the unreliable CoAP transport layer.

#### 7.3.5.6.1.2 CoAP transport layer confirmations

DLMS/COSEM CoAP TL confirmations are supported to provide explicit transfer confirmation of the receipt of a transferred APDU by the receiving DLMS/COSEM entity when no DLMS/COSEM AL or AP response is otherwise returned making such a confirmation.

DLMS/COSEM CoAP TL confirmations are required for reliable data push operation using unconfirmed DataNotification with push\_operation\_method (1). See DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.8.2.2.11).

DLMS/COSEM CoAP TL confirmations are supported by the CoAP wrapper layer for APDUs that are provided with Response\_Mode = UNCONFIRMED and Transport\_Mode = RELIABLE in the CoAP-DATA.request primitive.

The process is as follows:

- An APDU provided to the CoAP wrapper in the CoAP-DATA.request primitive with Response\_Mode = UNCONFIRMED shall be transferred by the CoAP wrapper within a new locally originating CoAP request/response context in a CWPDU with the CoAP Wrapper Response Mode set to 1 (WRM = 1). This instructs the receiving CoAP wrapper not to wait for a DLMS AL response or DLMS AP response to be returned;
- A receiving CoAP wrapper shall upon successful delivery of a received embedded APDU to the DLMS AL, when the APDU was received in a CWPDU with WRM = 1 over the reliable CoAP messaging layer<sup>1</sup>, return an empty CWPDU as a successful response to the sending CoAP wrapper entity;
- Error handling for a CWPDU received with WRM = 1 follows the general error handling as described above.

NOTE 1: For APDUs received with WRM = 1 over the unreliable CoAP messaging layer, the CoAP wrapper simply releases the context upon receipt of the CWPDU.

For further information on the CoAP wrapper state machine and the handling of the WRM field, see 7.3.5.7.3.5.

A message sequence flow depicting transfer of an unconfirmed DLMS DataNotification with return of a DLMS/COSEM CoAP TL confirmation is given in 7.3.6.3.2.

#### 7.3.5.7 CoAP wrapper state machine

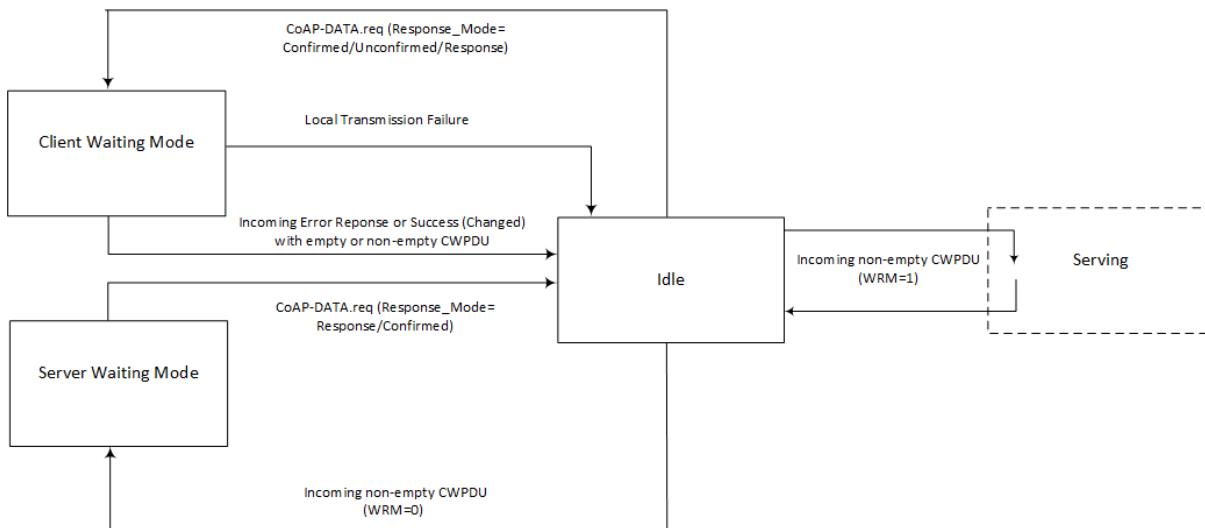
##### 7.3.5.7.1 CoAP wrapper request-response context states

A CoAP wrapper request/response context may exist in one of four states:

124/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

- *Idle*: Closed state with no associated state and no corresponding CoAP request/response context in the CoAP request/response layer
- *Client Waiting Mode*: Open, waiting state with a corresponding CoAP request/response context in the CoAP request/response layer initiated by the CoAP wrapper itself.
- *Server Waiting Mode*: Open, waiting state with a corresponding CoAP request/response context in the CoAP request/response layer initiated by a remote CoAP wrapper.
- *Serving*: Open, transient state with a corresponding CoAP request/response context in the CoAP request/response layer initiated by a remote CoAP wrapper in which an incoming non-empty CWPDU with CoAP wrapper Response Mode (WRM) = 1 is being served.

The high level state machine of the CoAP wrapper is shown in Figure 49.



**Figure 49 – High-level state transition diagram for the CoAP wrapper layer**

The individual state transitions are described in detail in 7.3.5.7.3.

#### 7.3.5.7.2 CoAP DLMS/COSEM wrapper request/response context

The state parameters that shall be maintained in the three open states CoAP wrapper request/response states are listed in Table 10.

**Table 10 – CoAP wrapper request/response context parameters**

State parameters	Client Waiting Mode	Server Waiting Mode	Serving
CoAP request/response context reference	X	X	X
Transport_Mode	X	X	X
Request_ID	(X)	(-)	(-)
Local_SAP	X	X	X
Remote_SAP	X	X	X
Local_IP_address	(X)	X	X
Local_Port	(X)	X	X
Remote_IP_address	X	X	X
Remote_Port	X	X	X

State parameters	Client Waiting Mode	Server Waiting Mode	Serving
X: The state parameter is always available.			
(X): The state parameter is conditionally available, i.e., when it is present in the CoAP-DATA.request APDU.			
(-) Not available			

With the exception of the CoAP request/response context reference, the parameters maintained in the Client Waiting Mode state are taken from the service parameters of the CoAP-DATA.request service primitive that resulted in the creation of the CoAP wrapper request/response context in this state. See Figure 49. The exact realization of and provisioning of the CoAP request/response context reference at the application programming interface provided by a CoAP protocol layer is an implementation decision.

The parameters maintained in the Server Waiting Mode and in the Serving state are taken from the lower CoAP protocol layers and from the CWPDU header of the incoming CWPDU that resulted in the creation of the CoAP wrapper request/response context in this state. See Figure 49. This includes the “CoAP request/response context reference” provided at the application programming interface.

#### 7.3.5.7.2.1 Match of CoAP-DATA.request invocation to a CoAP wrapper request/response context

A CoAP-DATA.request invocation by the DLMS AL is specified to match an existing CoAP wrapper request/response context in Server Waiting Mode when the Transport\_Mode, the Local\_SAP, the Remote\_SAP, the Local\_IP\_address, the Local\_port, the Remote\_IP\_address and the Remote\_Port specified in the CoAP-DATA.request invocation matches the parameters of the CoAP wrapper request/response context.

Such matching is the trigger of the Server Waiting Mode to Idle state transition. See Figure 49. For further details of the state transition see 7.3.5.7.3.

#### 7.3.5.7.2.2 Match of incoming CWPDU or CoAP layer failure indication to a CoAP wrapper request/response context

A CoAP response, an error response or a CWPDU, or a CoAP layer failure indication received from the CoAP request/response layer is matched to a CoAP wrapper request/response context in Client Waiting Mode through the “CoAP request/response context reference” provided.

Such a match provides the trigger of the Client Waiting Mode to Idle state transition. See Figure 49. For further details of the state transitions see 7.3.5.7.3.

#### 7.3.5.7.3 State transitions

##### 7.3.5.7.3.1 Idle to Client Waiting Mode Transition

This transition occurs in the following situations:

- 1) A CoAP-DATA.request invocation with Response\_Mode = UNCONFIRMED is received;
- 2) When no matching CoAP wrapper context in Server Waiting Mode state is found<sup>1</sup> for a received CoAP-DATA.request invocation from the DLMS AL with Response\_Mode = CONFIRMED;
- 3) When no matching CoAP wrapper context in Server Waiting Mode state is found<sup>1</sup> for a received CoAP-DATA.request invocation from the DLMS AL with Response\_Mode = RESPONSE.

NOTE 1 Only if combined CoAP client-server endpoints are used may there potentially be a matching CoAP wrapper request/response context in Server Waiting Mode state.

Case 1): A CoAP-DATA.request invocation by the DLMS AL with Response\_Mode = UNCONFIRMED is served towards a CoAP request/response wrapper context in Idle state and results in the creation of a new CoAP wrapper request/response context in Client Waiting Mode state. The Client Waiting Mode state parameters, other than the CoAP request/response context reference, are populated from the service parameters of the CoAP-DATA.request invocation. The CWPDU header is created with WRM = 1 and contains the LSAP and the RSAP. The DATA field carries the APDU. The CWPDU is

passed to the CoAP request/response layer where the corresponding CoAP request/response context is created, and the CWPDU is sent to the peer. The URI-Path shall be the same as in the Remote\_Path in the CoAP-DATA.request service primitive. See the procedure in Figure 11.

Case 2): The process is the same as in Case 1, except the CoAP-DATA.request service primitive is invoked with Response\_Mode = CONFIRMED and in the CWPDU header the WRM is set to 0. See the procedure in Figure 11.

Case 3): The process is the same as in Case 1, except that the CoAP-DATA.request service primitive is invoked with Response\_Mode = RESPONSE and the URI-Path is taken from the relevant CoAP setup object. See the procedure shown in Figure 11. This transition process is required to support DLMS GBT streaming in the DLMS/COSEM CoAP communication profile. For further details see message sequence diagrams in 7.3.6.6.

#### **7.3.5.7.3.2 Idle to Server Waiting Mode Transition**

This transition occurs when the DLMS/COSEM CoAP wrapper receives from the remote peer a well-formed, non-empty CWPDU with WRM = 0. The Server Waiting Mode state parameters are populated from the CWPDU header fields and from the information provided from the CoAP request/response layer (The CoAP request/response context reference, Transport\_Mode, local and remote IP addresses and ports). See the procedure shown in Figure 12. The APDU received is passed to the DLMS/COSEM AL using the CoAP-DATA.indication service primitive.

#### **7.3.5.7.3.3 Server Waiting Mode to Idle Transition**

This transition occurs in the following situations:

- 1) A matching CoAP wrapper context in Server Waiting Mode state is found for a received CoAP-DATA.request invocation from the DLMS AL with Response\_Mode = RESPONSE.
- 2) A matching CoAP wrapper context in Server Waiting Mode state is found<sup>1</sup> for a received CoAP-DATA.request invocation from the DLMS AL with Response\_Mode = CONFIRMED.

NOTE 1: This can only happen if combined CoAP client-server endpoints are used in both ends.

Case 1): The CoAP-DATA.request invocation from the DLMS AL with Response\_Mode = RESPONSE is served towards the peer through the matching CoAP wrapper request/response context in Server Waiting Mode state. The Server Waiting Mode state is left once the CoAP-DATA.request has been served towards the CoAP request/response layer for transmission of the given APDU within the existing CoAP request/response context. See the procedure shown in Figure 11.

Case 2): The process is the same as in Case 1, except that the CoAP-DATA.request service primitive is invoked with Response\_Mode = CONFIRMED. See the procedure in Figure 11. This transition process is required in order to support DLMS GBT streaming in the DLMS/COSEM CoAP communication profile with optimal piggybacking in both directions. For further details see message sequence diagrams in 7.3.6.6.

#### **7.3.5.7.3.4 Client Waiting Mode to Idle Transition**

This transition occurs when a response is returned form the CoAP request/response layer either in form of an error response or in form of an empty or non-empty CWPDU returned from the remote side or in form of a CoAP transmission failure. The Client Waiting Mode state is left once the APDU payload of a received non-empty CWPDU has been provided to the DLMS AL using the CoAP-DATA.indication primitive or when the resulting DLMS/COSEM CoAP TL confirmation or failure indication has been provided to the DLMS AL using the CoAP-DATA.confirm primitive, if supported. The response (error response or CWPDU) received from the CoAP request/response layer is matched to the CoAP wrapper request/response context through the CoAP request/response context reference provided. See the procedure shown in Figure 12.

NOTE A CoAP wrapper request/response context in Client Waiting Mode state created to serve a CoAP-DATA.request invocation with Transport\_Mode = UNRELIABLE and Response\_Mode = UNCONFIRMED (i.e., a CoAP wrapper request/response context associated with Transport\_Mode = UNRELIABLE created to serve the transfer of an unconfirmed DLMS AL service invocation or the transfer of a GBT APDU for which no response is expected) may be returned to idle already once the associated APDU has been delivered to the CoAP request/response layer for transfer, or it may be kept to allow for optional return of DLMS/COSEM failure indications.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	127/614
-----------------------	------------	-----------------------	---------

### 7.3.5.7.3.5 Idle to Serving to Idle Transition

This transition occurs when a response when a well-formed non-empty CWPDU with WRM = 1, carried through a new CoAP request/response context initiated from a remote peer, is received from the CoAP request/response layer. The state is transient. The serving state is entered upon the receipt of the CWPDU and left when the APDU payload of the received CWPDU has been provided to the DLMS AL and, if reliable transport is used, that the return of an empty CWPDU through the incoming CoAP request/response context has been served towards the CoAP request/response layer. See the procedure in Figure 12 and for reference see 7.3.5.6.1.2.

### 7.3.5.7.4 CoAP-DATA.request invocation handling

The handling of an CoAP-DATA.request invocation, the formation and the transmission of the resulting CWPDU and the associated CoAP wrapper response/request context state transition is specified in Figure 50.

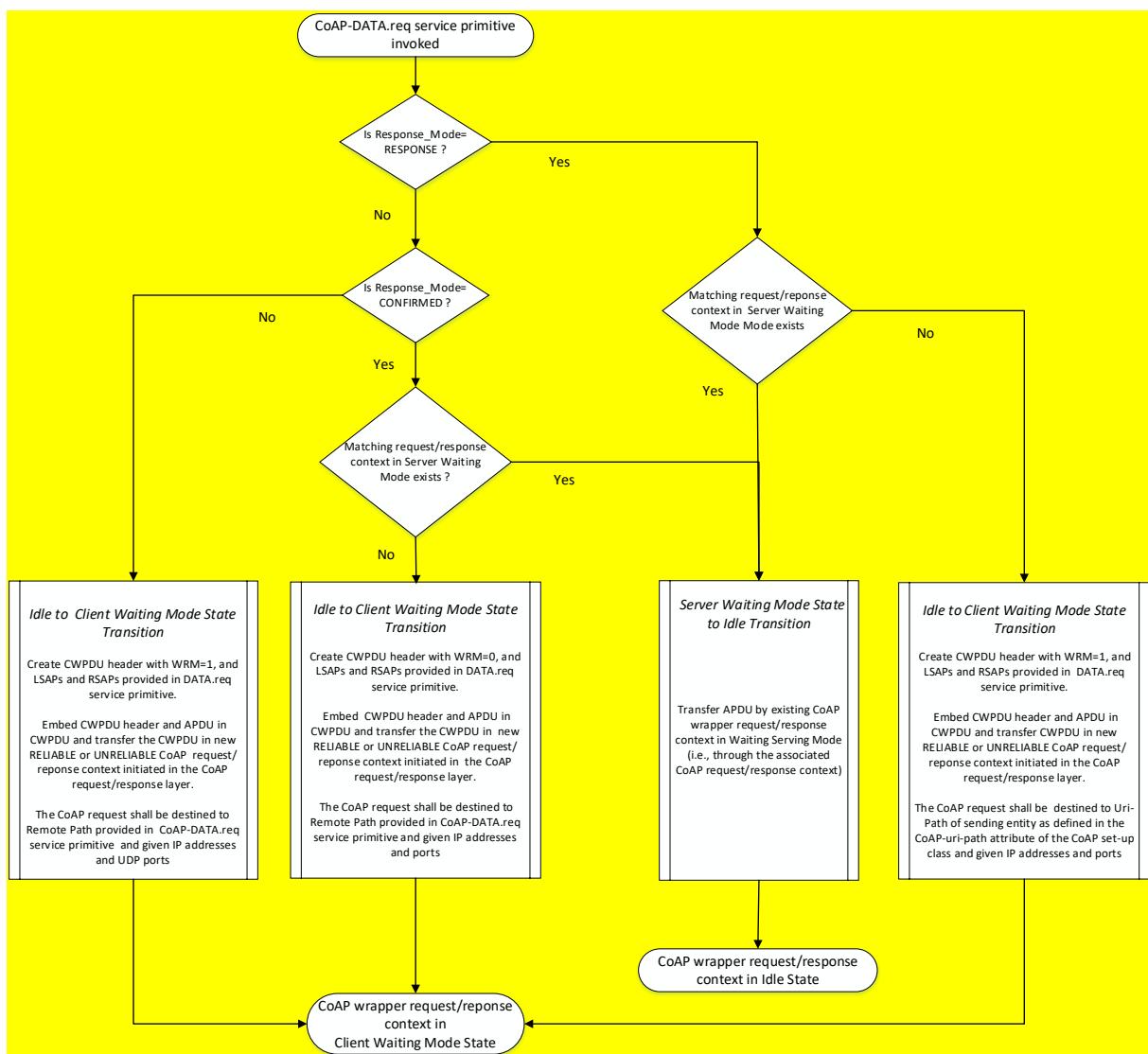
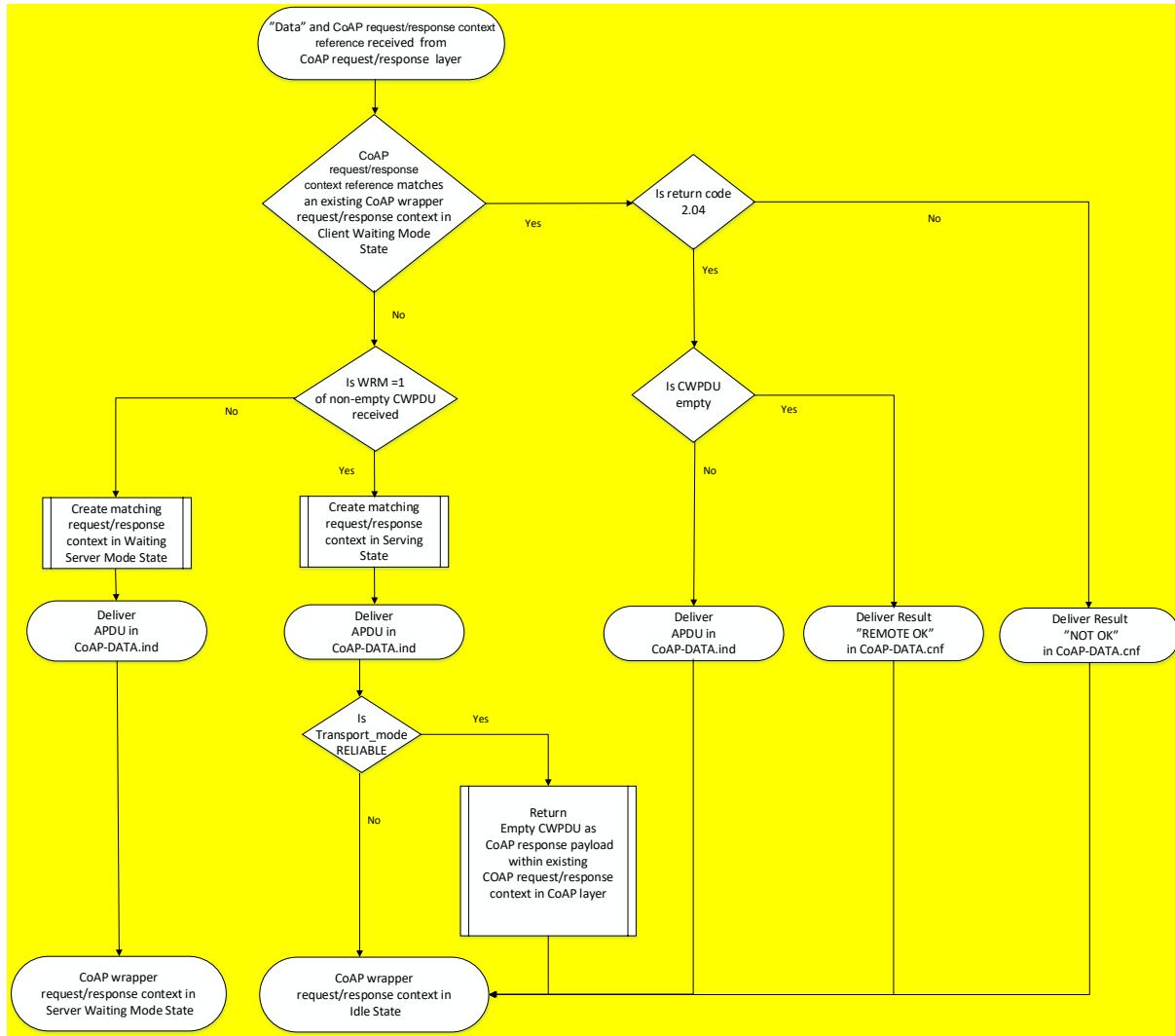


Figure 50 – CoAP-DATA.request invocation handling

### 7.3.5.7.5 Handling of incoming CWPDU or CoAP layer transmission failures

The handling of an incoming CWPDU or a CoAP layer transmission failure from the CoAP protocol layer, the propagation of the received information to the DLMS AL and the corresponding CoAP wrapper request/response context state transition is shown in Figure 51.

A CoAP transmission failure may come in form either of an error response or a CoAP request/response layer transmission failure. See 7.3.5.5.



NOTE The receipt of an error response or a CoAP request/response layer transmission failure is only optionally communicated in the CoAP-DATA.cnf primitive (Result "NOT OK") for the unreliable DLMS/COSEM CoAP TL.

**Figure 51 – Handling of incoming CWPDU or CoAP layer transmission failure**

### 7.3.5.7.6 Garbage collection

The method for how and when to clean out request/response contexts in the CoAP wrapper layer is implementation specific.

## 7.3.6 DLMS/COSEM CoAP TL Data Transfers

### 7.3.6.1 General

The message sequence flow examples in this clause illustrate the usage of the DLMS/COSEM CoAP TL DATA primitives by the DLMS AL and the resulting DLMS/COSEM CoAP TL transport operation for various DLMS application layer service invocations with and without usage of DLMS GBT.

The usage of combined CoAP client and server endpoints are shown in the examples whenever they are a prerequisite for the operation. The Local\_Port, the Local\_IP\_Address, the Remote\_Port and the Remote\_IP\_Address information are not shown in the examples.

### 7.3.6.2 General transfer of confirmed DLMS/COSEM AL service requests

The transfer of confirmed DLMS AL service request, including a confirmed DataNotification service request through the reliable and the unreliable DLMS/COSEM CoAP TL, is shown in Figure 52.

The flows shown in Figure 13 depict the general situation where a confirmed DLMS AL service request through the DLMS/COSEM CoAP TL does not require either CoAP block transfer operation or DLMS GBT.

The flow over reliable CoAP TL in Figure 13 depicts the situation where no retransmissions are required at the CoAP protocol layer and where the response APDU is combined with the CoAP acknowledgement message through piggybacking.

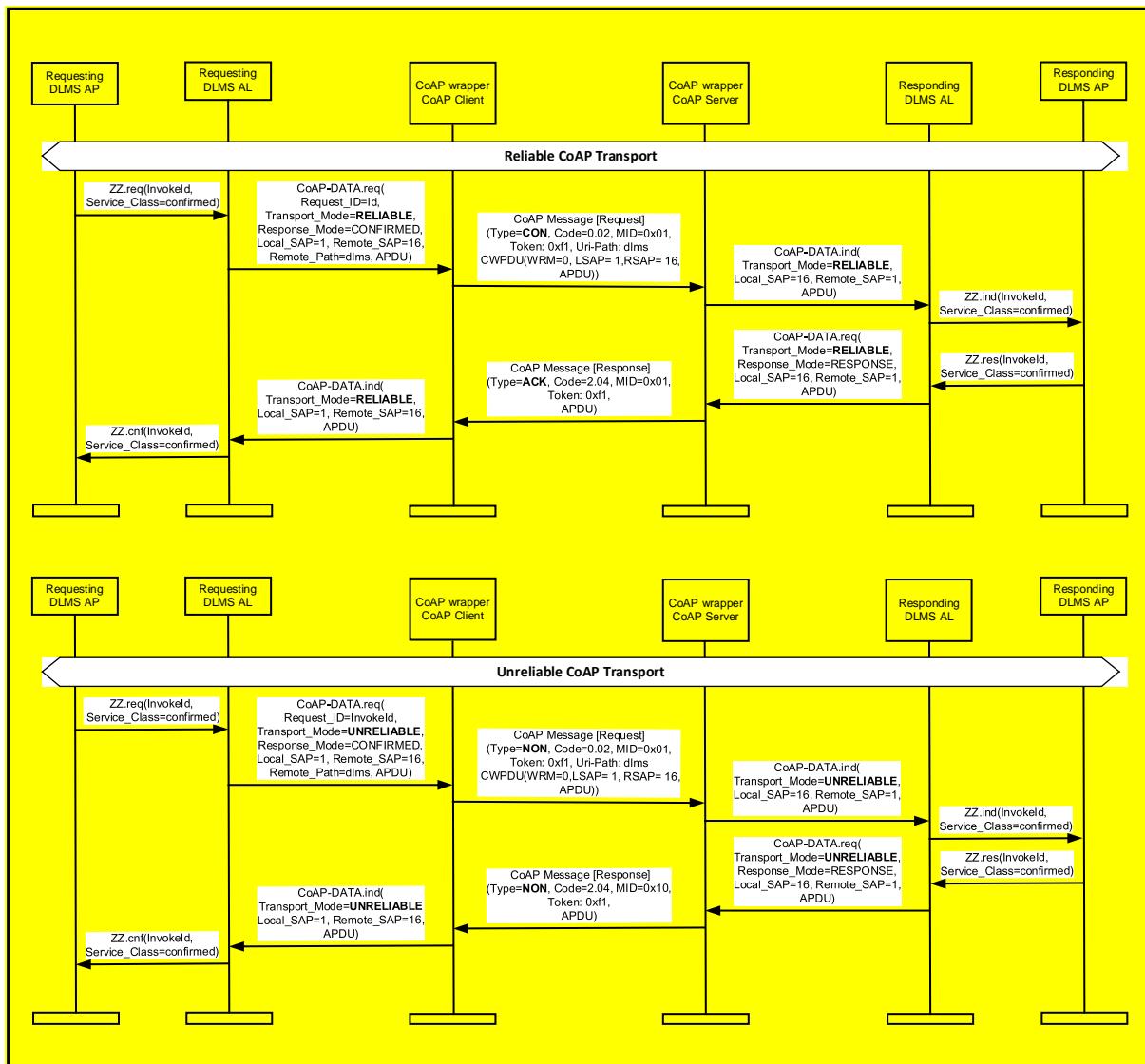


Figure 52 – Confirmed DLMS/COSEM AL service request through CoAP TL

**7.3.6.3 Reliable DLMS/COSEM CoAP TL operation****7.3.6.3.1 Transfer of confirmed DLMS/COSEM AL service requests over reliable CoAP TL**

The transfer of confirmed DLMS AL service request invocation, including a confirmed DataNotification service request through the reliable DLMS/COSEM CoAP TL with and without piggybacking of the response APDU with the CoAP acknowledgement message is shown in **Figure 53**.

Some simple DLMS/COSEM CoAP TL loss recovery examples are shown in Figure 54.

In all cases note the following;

- Whether piggybacked or not, the CoAP response is sent from the CoAP server endpoint (IP address and CoAP server UDP port) to the CoAP client endpoint (IP address and CoAP client UDP port);
- The CoAP acknowledgement message is used to control the CoAP retransmission operation and to prevent spurious retransmissions. The receipt of the CoAP acknowledgement message is not directly exposed outside of the CoAP messaging layer. I.e., it is not assumed to be exposed at the interface towards the CoAP wrapper and it is not assumed to be exposed towards the DLMS/COSEM AL.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	131/614
-----------------------	------------	-----------------------	---------

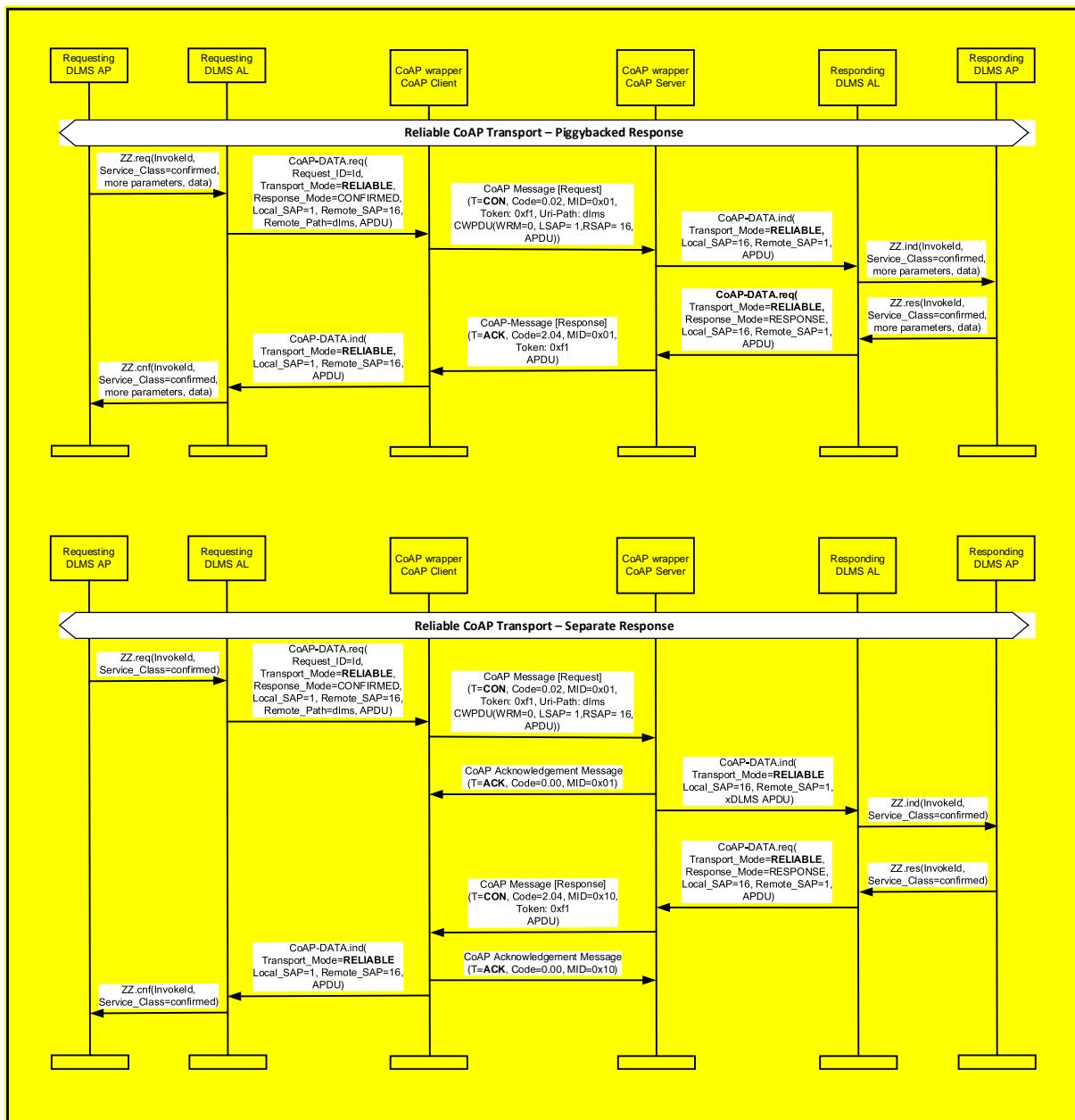


Figure 53 – Piggybacked and separate response handling with reliable CoAP TL

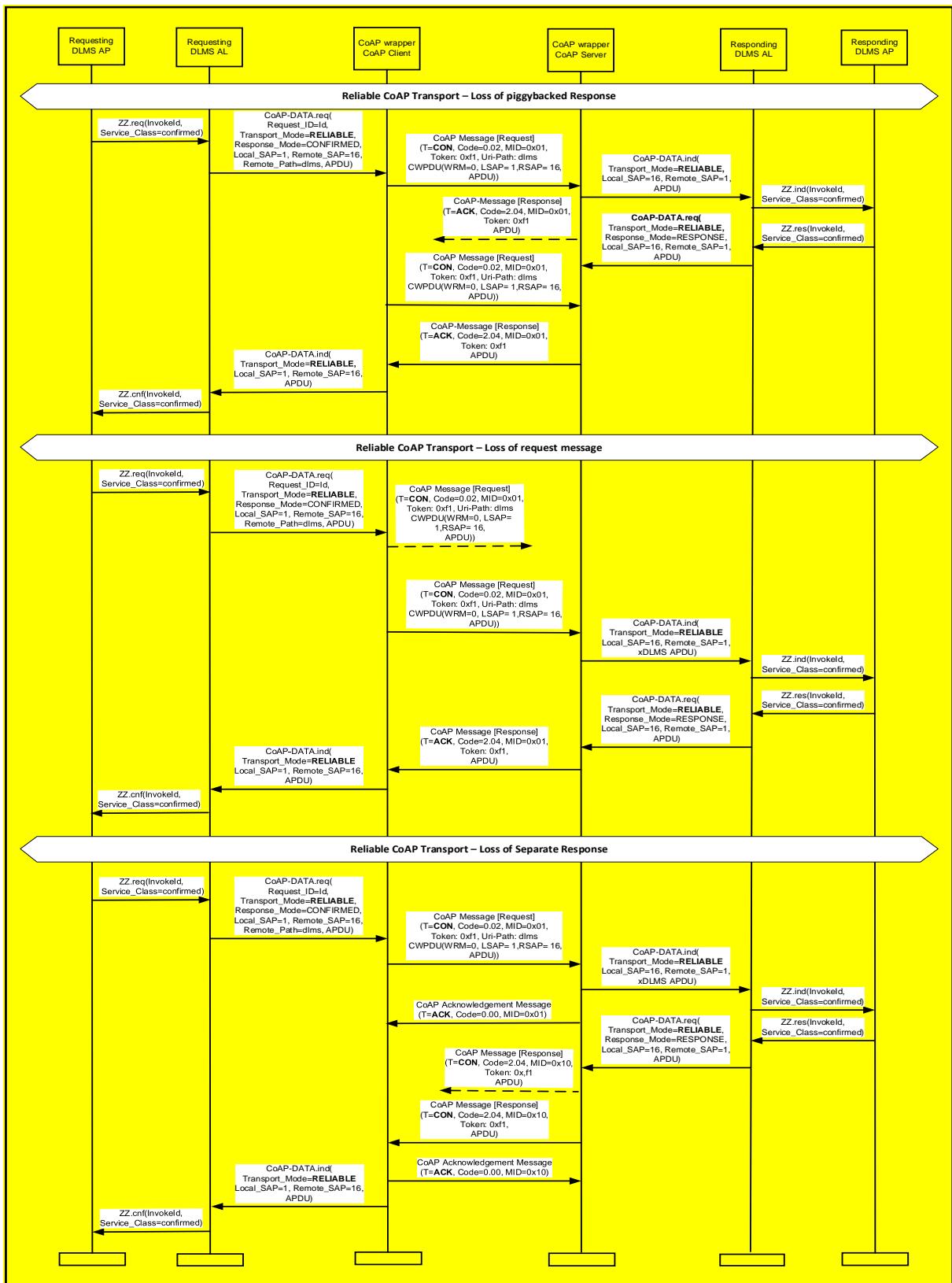
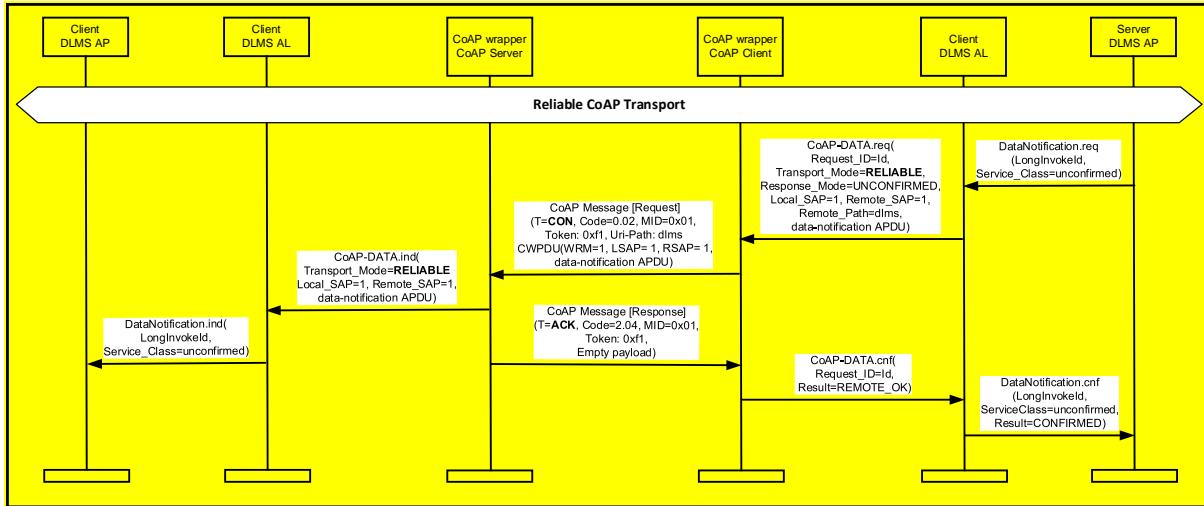


Figure 54 – Loss Recovery of the reliable DLMS/COSEM CoAP TL

### 7.3.6.3.2 Transfer of unconfirmed DLMS/COSEM AL service requests over reliable CoAP TL

The transfer of unconfirmed DLMS AL service request invocation, here shown by an unconfirmed DataNotification service request, through the reliable DLMS/COSEM CoAP TL with piggybacking of the DLMS/COSEM CoAP TL confirmation (the empty CWPDU, or empty response payload) with the CoAP acknowledgement message is shown Figure 55.



**Figure 55 – Unconfirmed DataNotification through reliable CoAP TL with DLMS/COSEM CoAP TL confirmation**

The DLMS AL will have to keep state of the Request\_ID value provided in the CoAP-DATA.request primitive and the LongInvokeld of the DataNotification invocation so that it may return the LongInvokeld in the DataNotification.cnf service primitive (see GB 10 9.4.6.7). This may be achieved by using the LongInvokeld as the Request\_ID in this situation.

### 7.3.6.4 Unreliable DLMS/COSEM CoAP TL operation

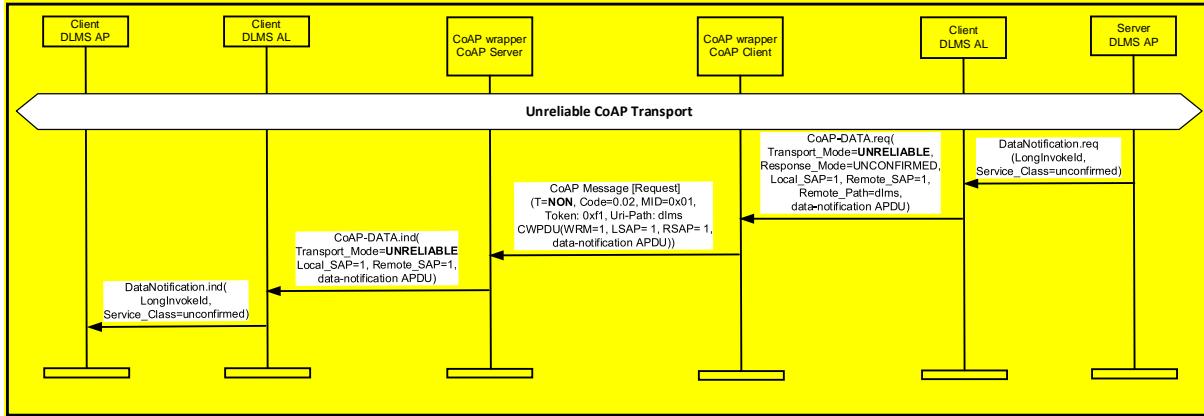
#### 7.3.6.4.1 General

With unreliable DLMS/COSEM CoAP TL no reliability is provided from the CoAP messaging layer for the delivery of the CoAP request message to peer CoAP endpoint nor for the return of a CoAP response.

The mapping of a confirmed DLMS AL service request, including a confirmed DataNotification service request, over the unreliable CoAP TL is shown in Figure 52.

#### 7.3.6.4.2 Transfer of unconfirmed DLMS/COSEM AL service requests over unreliable CoAP TL

An unconfirmed DLMS application layer service request over the unreliable DLMS/COSEM CoAP TL is transferred with CoAP Wrapper Response Mode, WRM = 1. See Figure 56.



**Figure 56 – Unconfirmed DataNotification through unreliable CoAP TL**

Neither the sending CoAP wrapper nor the receiving CoAP wrapper need to maintain state to facilitate response return in this situation and may immediately release the request/response context after use.

#### 7.3.6.5 DLMS/COSEM CoAP Block Transfer operation

The CoAP Block Transfer mechanism provided within the DLMS/COSEM CoAP TL serves to transfer DLMS/COSEM APDUs of size larger than the MTU size but still smaller than the receiver\_max\_pdu\_size.

The basic operation of CoAP block transfer of the DLMS/COSEM CoAP TL in accordance with the CoAP Block transfer operation defined by RFC 7959 is shown in Figure 57, Figure 58, Figure 59, Figure 60 and Figure 61 below.

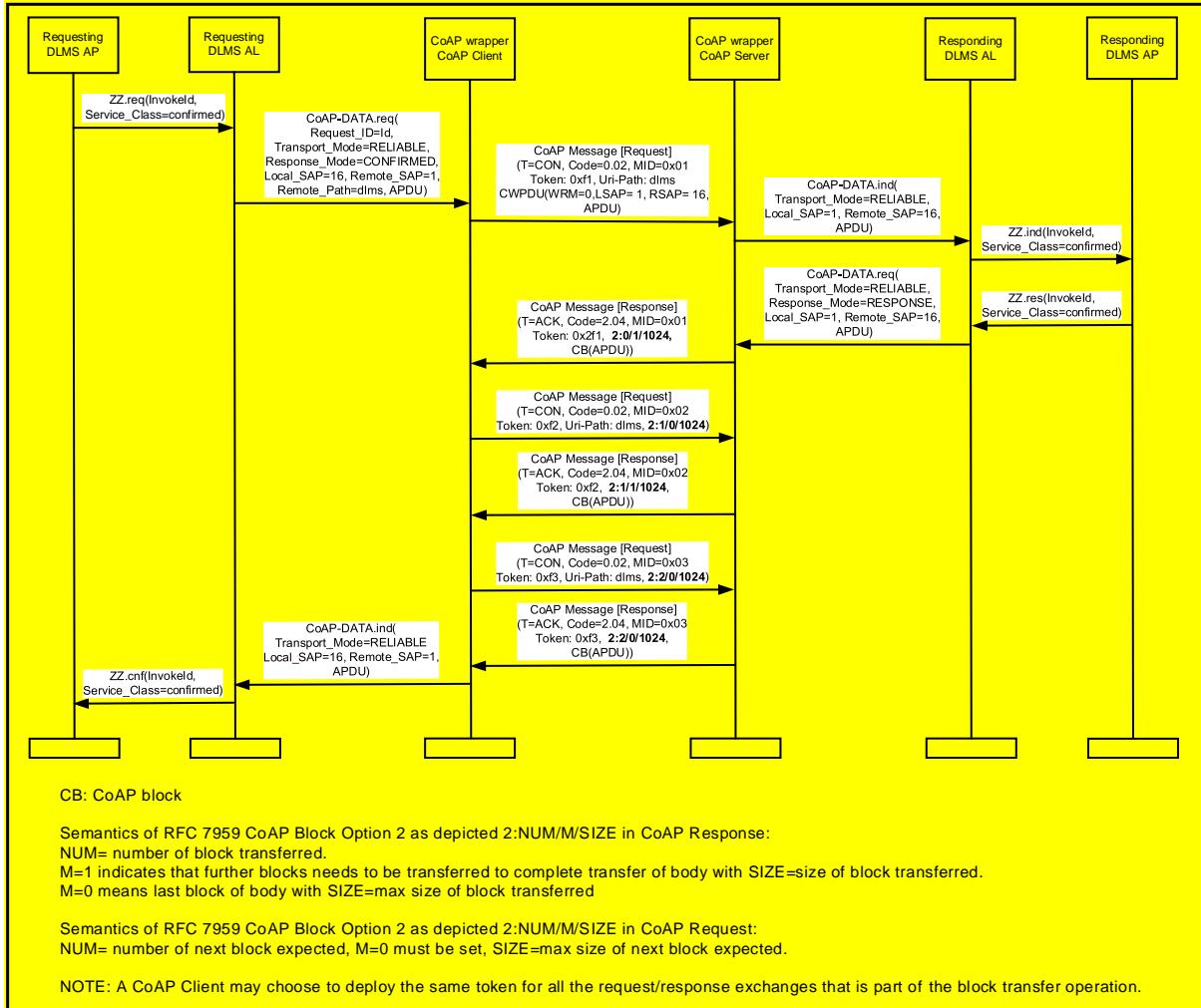
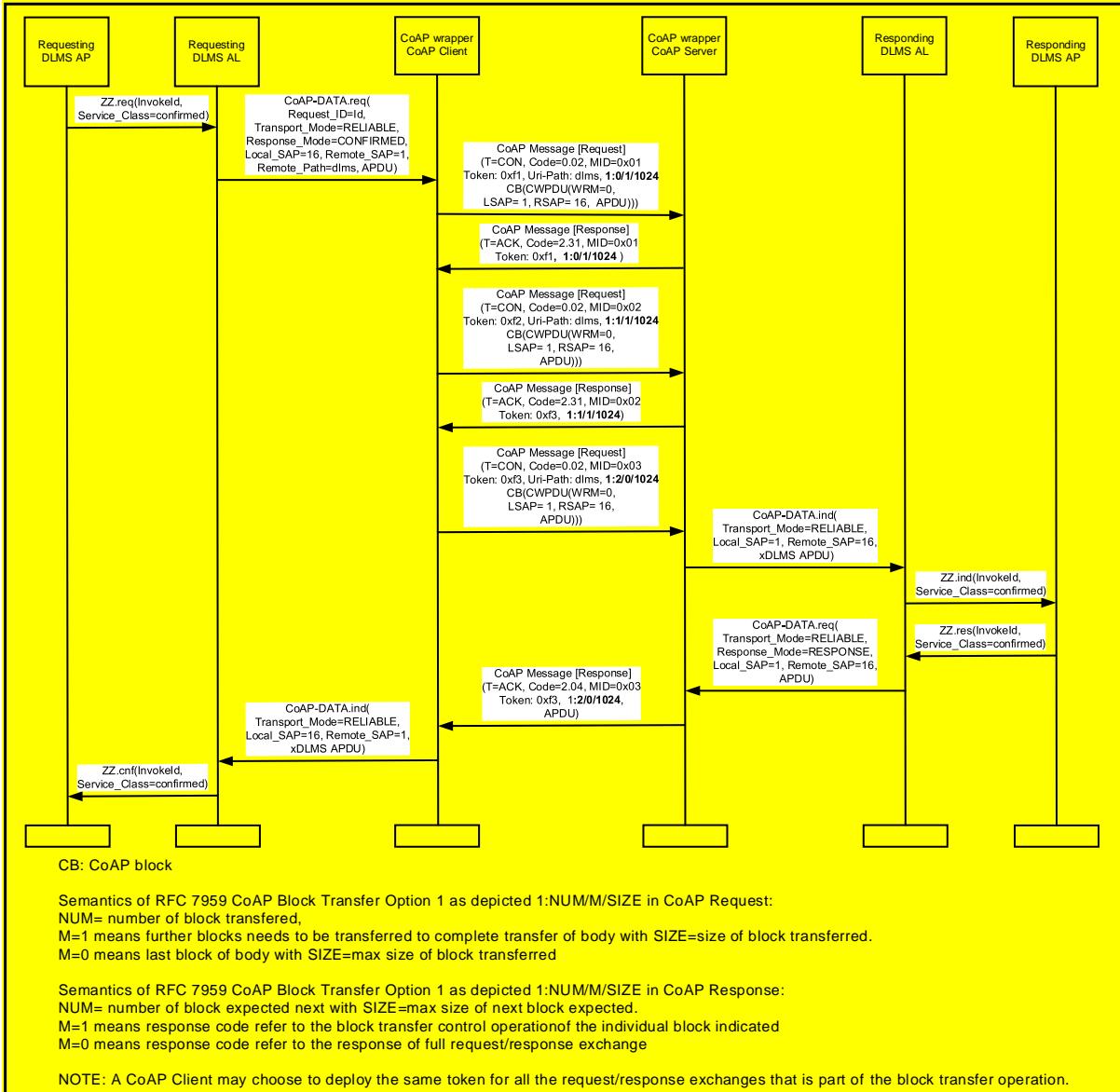
**Figure 57 – CoAP BT of a response APDU over reliable CoAP TL**

Figure 57 shows the CoAP block transfer operation for transfer of a response APDU of a confirmed DLMS service invocation through the reliable DLMS/COSEM CoAP TL in case the responses are returned piggybacked on the acknowledgment messages.

CoAP block transfer operation for transfer of a response APDU of a confirmed DLMS service invocation through the unreliable DLMS/COSEM CoAP TL will proceed in the same manner as the message sequence flow shown in Figure 57 with the difference that `Transport_Mode = UNRELIABLE` would be indicated in the `CoAP-DATA.req` primitives and the type of the CoAP messages would be “NON”.

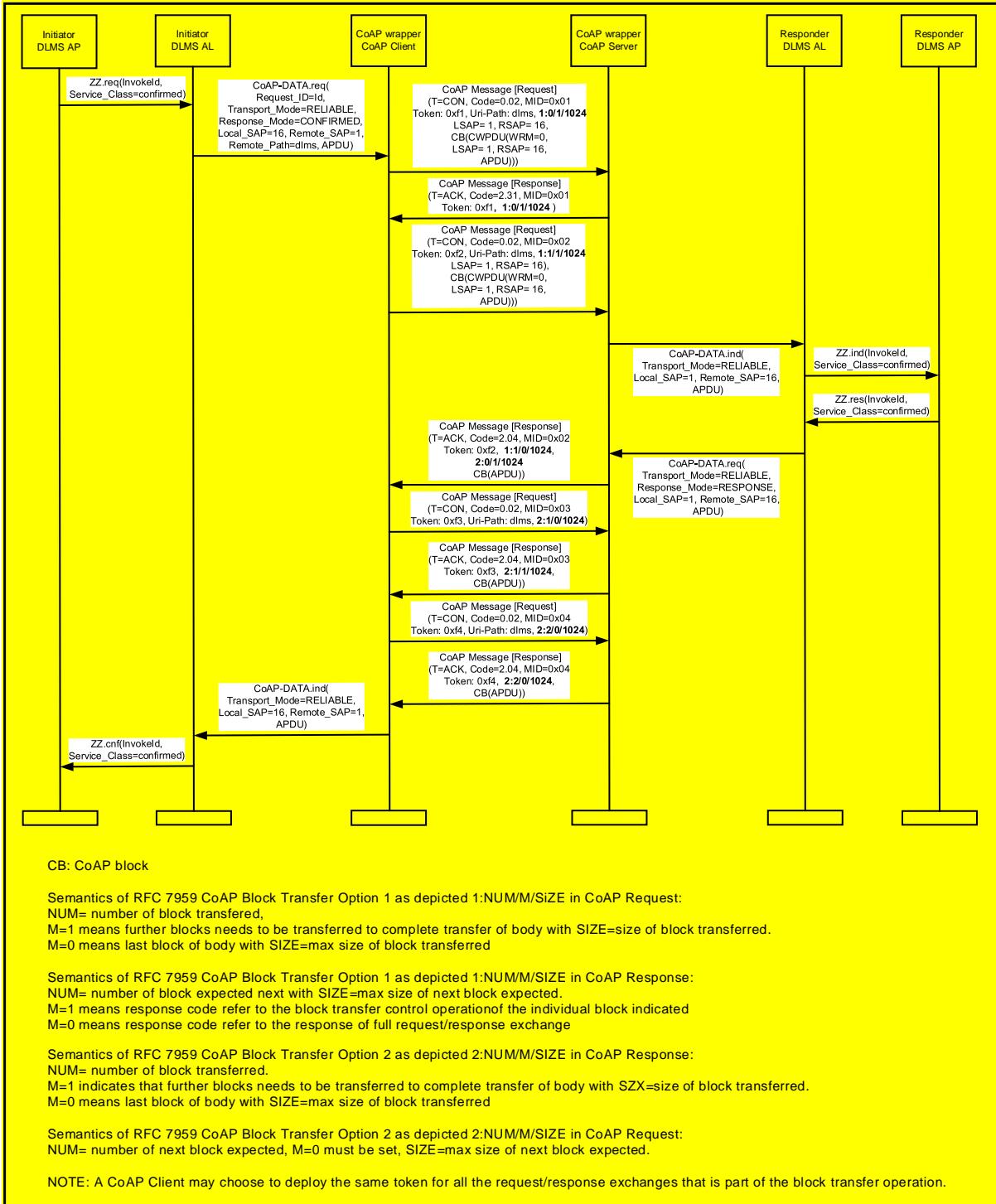
In case the also the request APDU exceeds the MTU size, the return of the response APDU proceeds through CoAP block transfer as shown in Figure 59.



**Figure 58 – CoAP BT of a request APDU over reliable CoAP TL**

Figure 58 shows the CoAP block transfer operation for transferring of an APDU carrying a confirmed service invocation request using the reliable DLMS/COSEM CoAP TL.

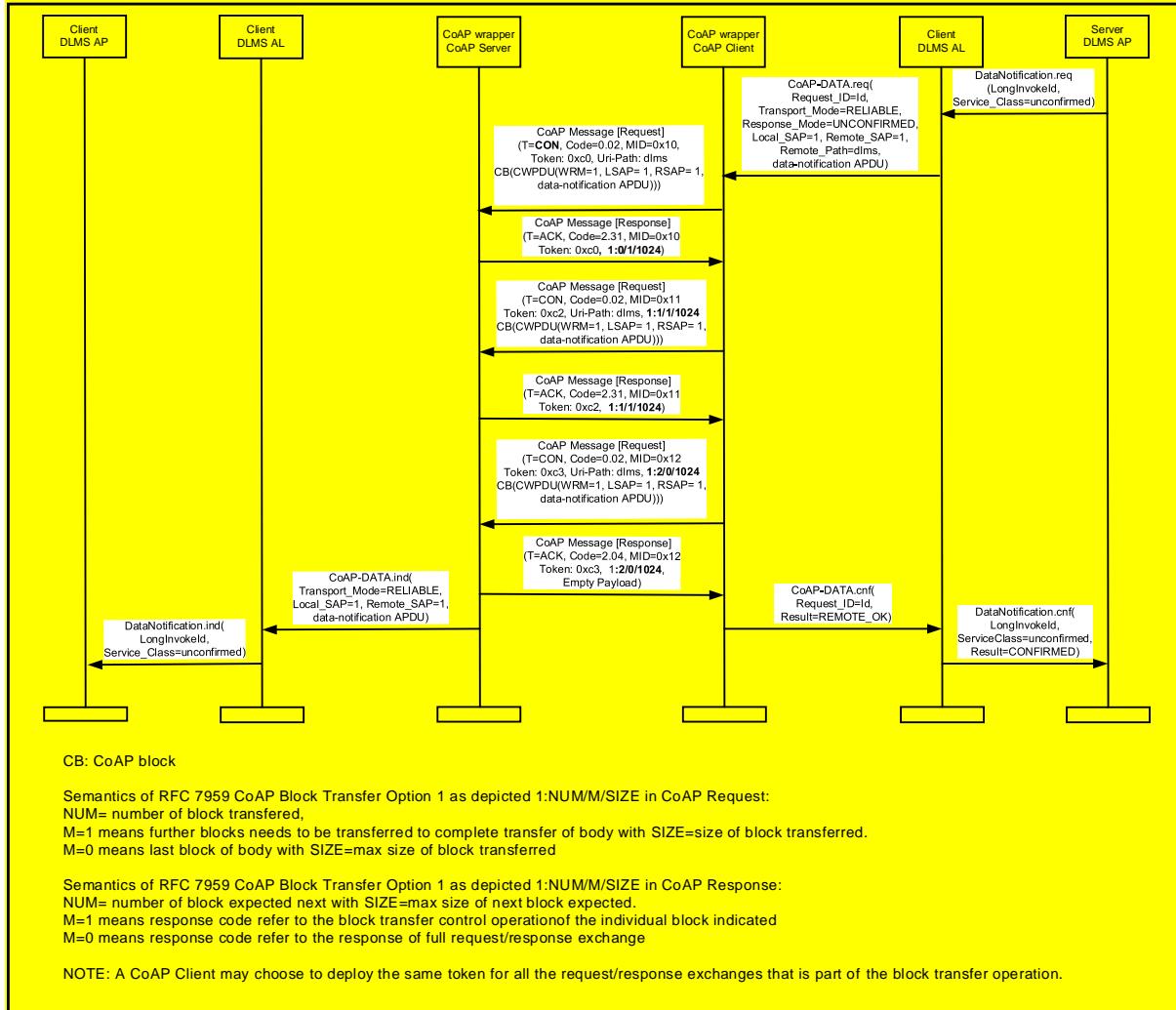
CoAP block transfer operation for transfer of a confirmed service invocation request using the unreliable DLMS/COSEM TL will proceed in the same manner as the message sequence flow shown in Figure 58 with the difference that Transport\_Mode = UNRELIABLE would be indicated in the CoAP-DATA.req primitives and the type of the CoAP messages would be "NON".



**Figure 59 – CoAP BT of request and response APDUs over reliable CoAP TL**

Figure 59 shows the CoAP block transfer operation for transfer of a confirmed service invocation through the reliable DLMS/COSEM CoAP TL where both request and response APDUs exceed the MTU size.

CoAP block transfer operation for transfer of a confirmed service invocation through the unreliable DLMS/COSEM CoAP TL where both request and response APDUs exceed the MTU size will proceed in the same manner as the message sequence flow shown in Figure 59 with the difference that Transport\_Mode = UNRELIABLE would be indicated in the CoAP-DATA.req primitives and the type of the CoAP messages would be "NON".



**Figure 60 – CoAP BT of an unconfirmed DataNotification over reliable CoAP TL**

Figure 60 shows CoAP block transfer operation for transfer of an unconfirmed DataNotification through the reliable DLMS/COSEM CoAP TL with DLMS/COSEM CoAP TL confirmation.

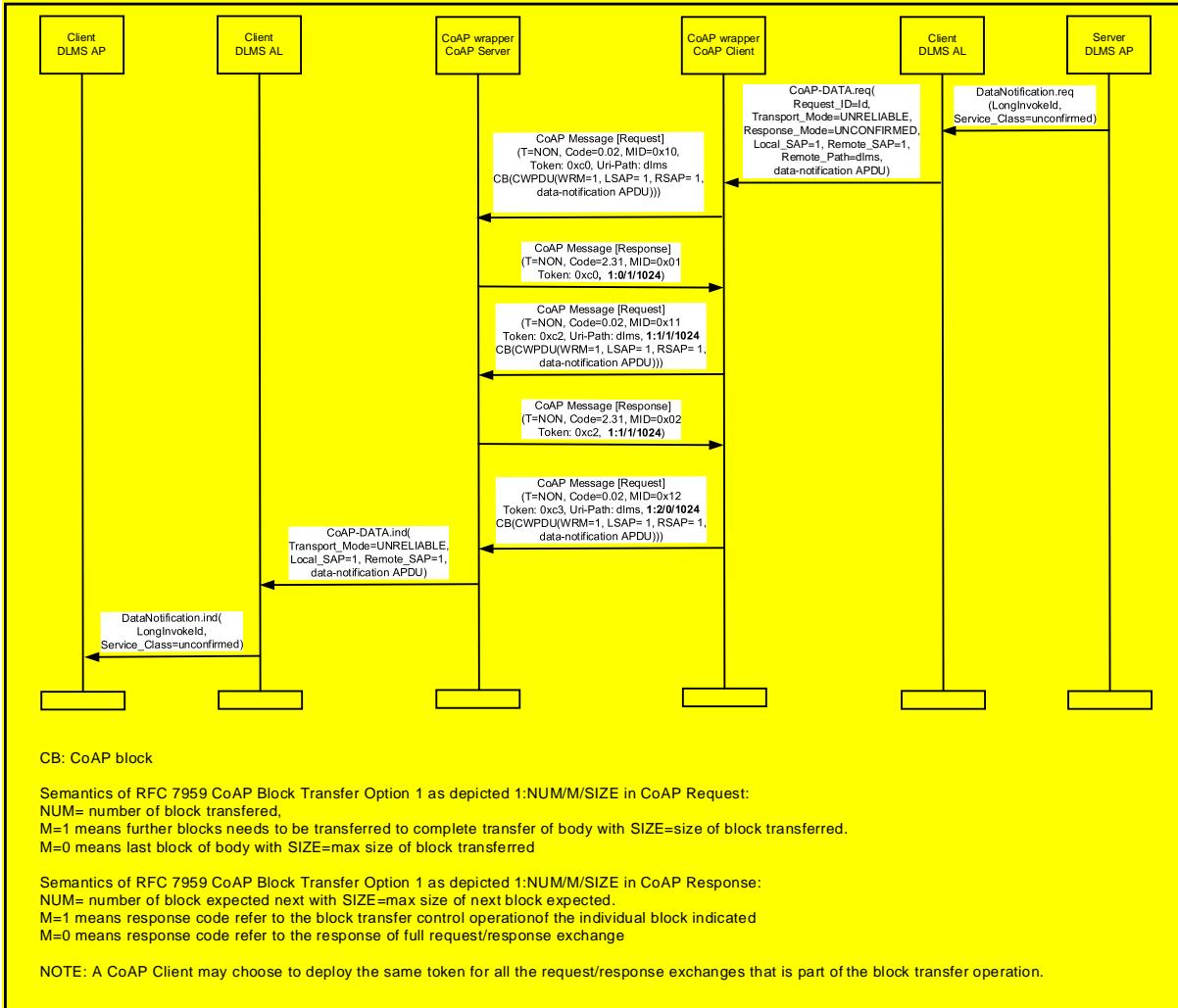
**Figure 61 – CoAP BT of an unconfirmed DataNotification over unreliable CoAP TL**

Figure 61 shows CoAP block transfer operation for transfer of an unconfirmed DataNotification through the unreliable DLMS/COSEM CoAP TL.

### 7.3.6.6 DLMS GBT operation over DLMS/COSEM CoAP TL

#### 7.3.6.6.1 General

The CoAP block transfer operation provided within the DLMS/COSEM CoAP TL serves to transfer large APDUs that exceed the MTU size without requiring IP layer fragmentation. The DLMS/COSEM general block transfer (GBT) mechanism can be supported atop the DLMS/COSEM CoAP TL for DLMS/COSEM layer block transfer segmentation into multiple GBT APDUs of large APDU payloads surpassing the receiver\_max\_pdu\_size.

DLMS/COSEM GBT may in some circumstances be preferred over CoAP block transfer. This may be controlled by constraining the receiver\_max\_pdu\_size to the APDU size that is desired to be transferred using CoAP block transfer or it may be controlled autonomously by the sending DLMS AL

For an illustration of the usage of GBT DLMS GBT in combination with CoAP BT within the DLMS/COSEM CoAP TL see 7.3.6.6.2.

For usage of DLMS/COSEM general block transfer (GBT) mechanism with streaming (with streaming window size larger than one) combined CoAP client and server endpoints shall be used within the DLMS/COSEM CoAP TL. See 7.3.6.6.3.

**7.3.6.6.2 CoAP block transfer in combination with GBT**

The combination of CoAP block transfer serving to transfer GBT APDUs of size receiver\_max\_pdu\_size exceeding the MTU size and DLMS GBT serving to transfer APDUs of size larger than the receiver\_max\_pdu\_size is shown in Figure 62.

The situation in Figure 62 illustrates GBT with streaming window equal to one 1. In situations with GBT with streaming window size equal to 1, usage combined CoAP client and server endpoints is not a prerequisite.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	141/614
-----------------------	------------	-----------------------	---------

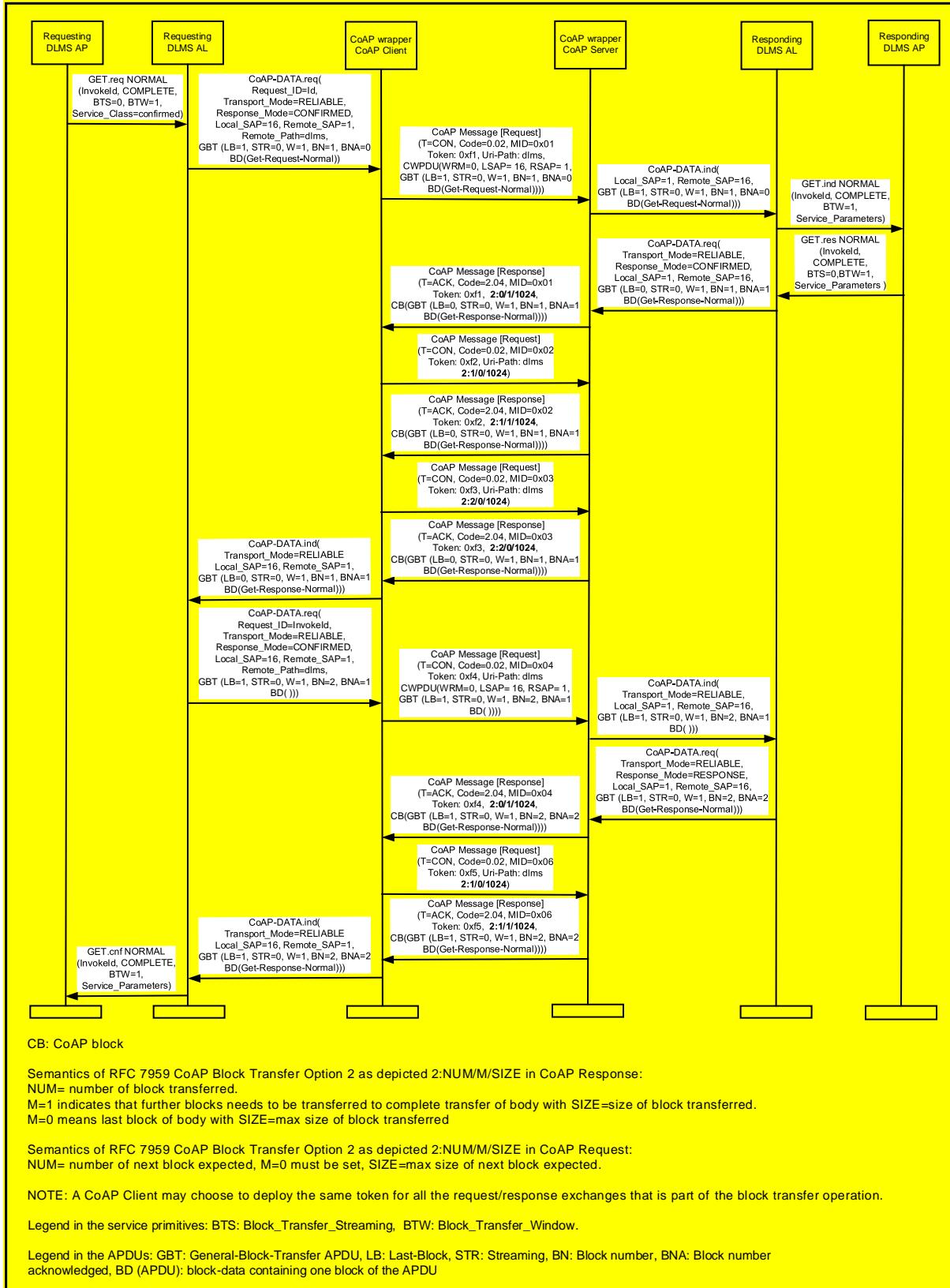


Figure 62 – CoAP BT in combination DLMS GBT for transfer of a large response APDU

**7.3.6.6.3 GBT with streaming over DLMS/COSEM CoAP TL****7.3.6.6.3.1 General**

A DLMS/COSEM implementation that supports GBT streaming over the DLMS/COSEM CoAP TL achieves this by having the DLMS/COSEM CoAP transport layer endpoints operate through combined CoAP client and server endpoints.

**7.3.6.6.3.2 Streaming of confirmed request APDUs****7.3.6.6.3.2.1 General**

For GBT transfer with streaming of a confirmed DLMS AL service request from a DLMS client to a DLMS server or of a confirmed DataNotification sent from a DLMS server to a DLMS client, the message communication flow consists of multiple GBT message transmissions for which no DLMS AL response messages are to be returned.

When streaming request GBT APDUs over DLMS/COSEM CoAP TL for which no response is assumed to be returned by GBT semantics, the Response\_Mode in the CoAP-DATA.request primitive is set to UNCONFIRMED by the DLMS AL. This will instruct the local CoAP wrapper to transmit the CoAP request message with WRM = 1 set and it will allow for the receiving CoAP wrapper to not await a response from the DLMS AL to be returned with the request/response context in question. See 7.3.5.7.4 and 7.3.5.7.5.

**7.3.6.6.3.2.2 Streaming of confirmed request APDUs over unreliable CoAP TL**

A confirmed GBT operation with streaming over unreliable CoAP transport is shown in Figure 63.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	143/614
-----------------------	------------	-----------------------	---------

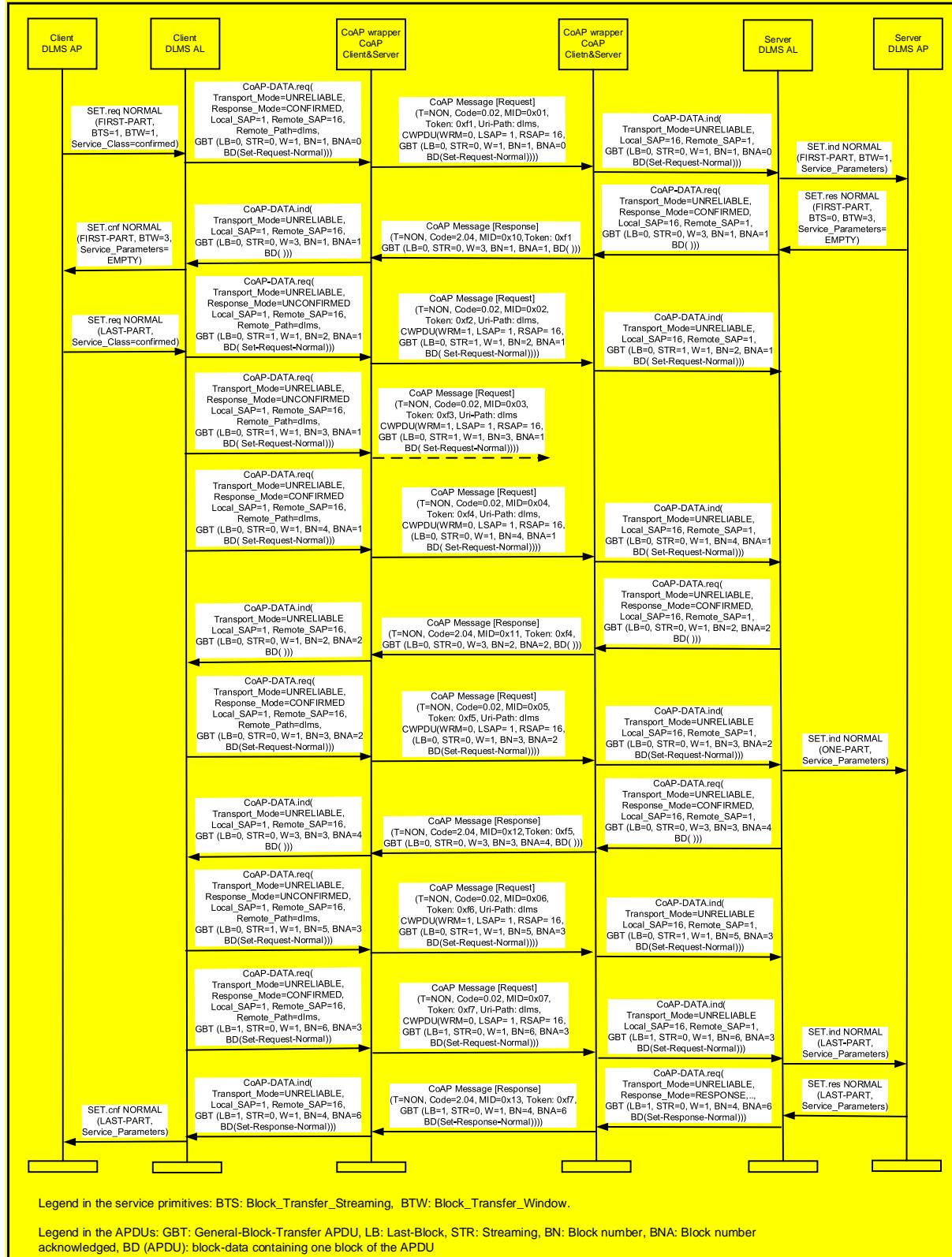
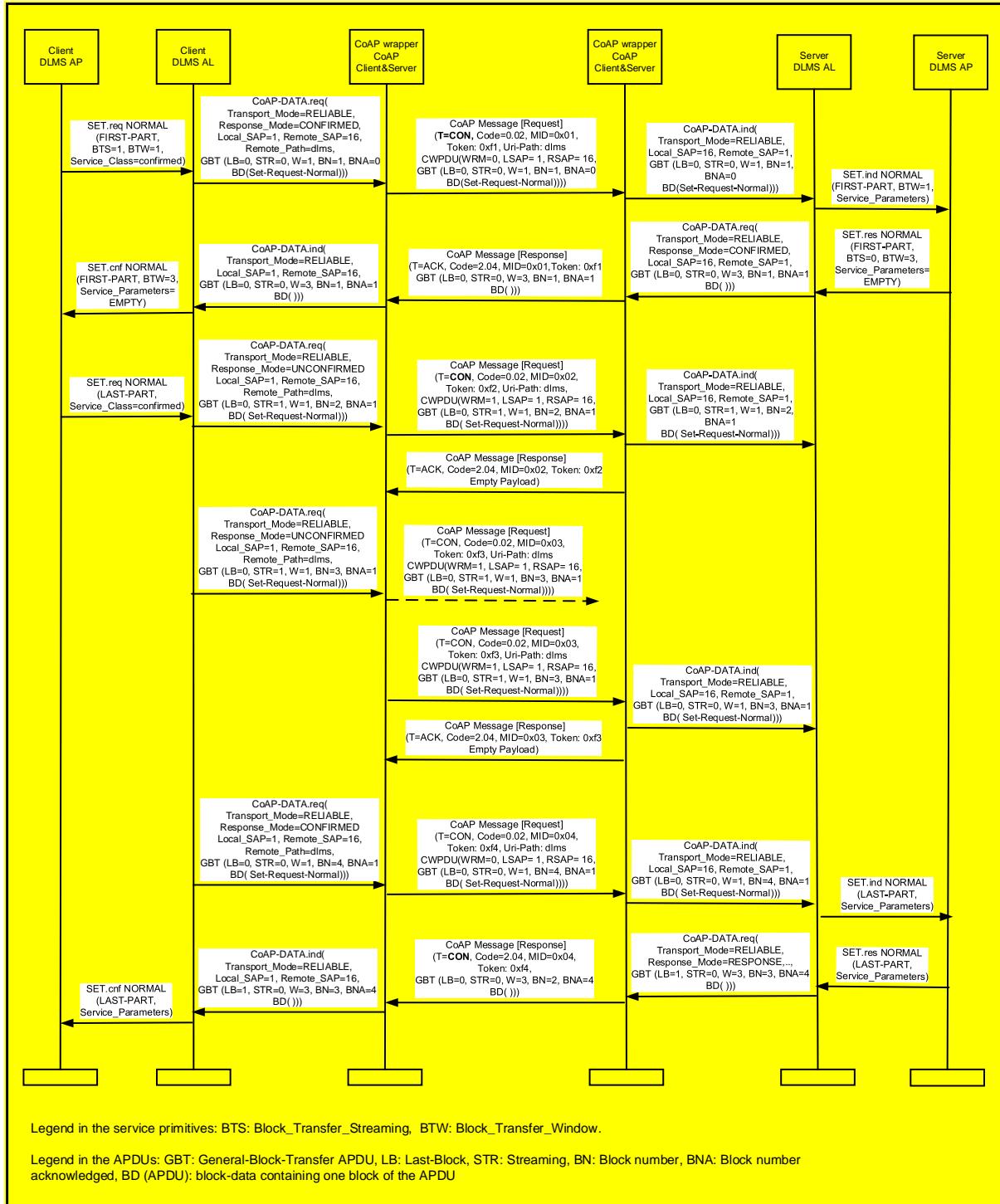


Figure 63 – SET service with GBT streaming over unreliable CoAP TL.

Figure 63 shows a SET service with GBT and streaming where the 3rd block send by the DLMS Client AL is lost and recovered by GBT. For details on the GBT process see Figure 148.

### 7.3.6.6.3.2.3 Streaming of confirmed request APDUs over reliable CoAP TL

A confirmed GBT operation with streaming over reliable CoAP transport is shown in Figure 64.



**Figure 64 – SET service with GBT streaming and loss recovery by reliable CoAP TL**

Figure 64 shows a SET service with GBT and streaming of block 2, 3 and 4 where the 3<sup>rd</sup> block sent by the DLMS Client AL is lost and recovered by the reliable DLMS/COSEM CoAP TL.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	145/614
-----------------------	------------	-----------------------	---------

The flow shown in Figure 64 assumes that all response APDUs are returned timely enough to the CoAP wrapper layer for these to be returned piggybacked with the CoAP acknowledgement message. Transmission of block 4 needs to await the loss recovery of block 3 by the DLMS/COSEM CoAP TL<sup>1</sup>.

Note 1: Assuming set of RFC 7252 protocol default NSTART = 1 in the CoAP protocol layer. See 7.3.5.4.3.3.

### 7.3.6.6.3.3 Streaming of response APDUs

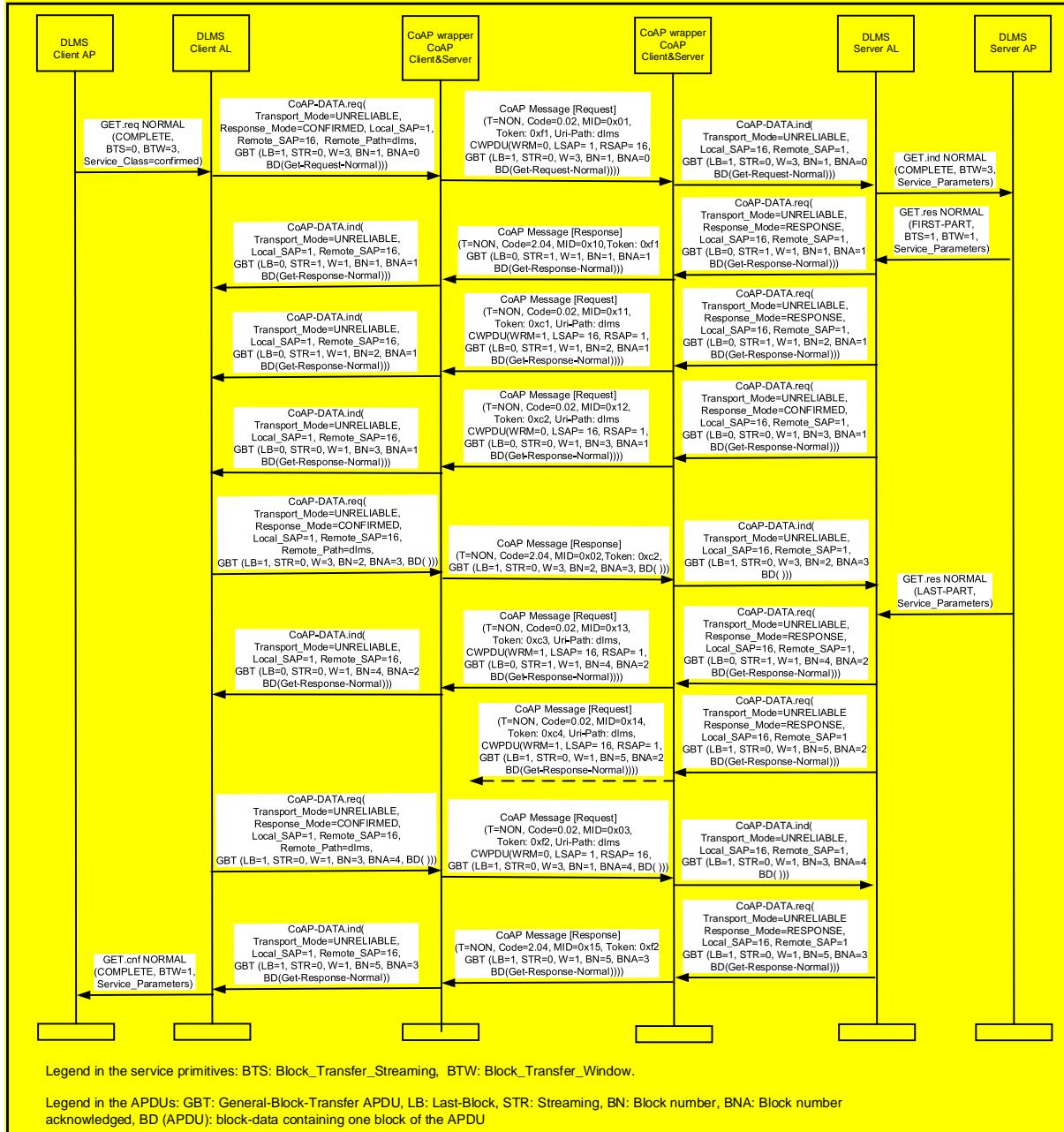
#### 7.3.6.6.3.3.1 General

For GBT transfer with streaming of a confirmed DLMS application layer service response, the message communication flow exhibits multiple consecutive GBT messages that shall be transferred from DLMS server to DLMS client as a result of one request.

When streaming response GBT APDUs over DLMS/COSEM CoAP TL for which no response is assumed to be returned by GBT streaming procedure perspective, the Response\_Mode in the CoAP-DATA.request primitive shall be set to RESPONSE by the DLMS AL. When streaming response GBT APDUs over DLMS/COSEM CoAP TL for which a response is expected from a GBT streaming procedure perspective, the Response\_Mode in the CoAP-DATA.request primitive shall be set to CONFIRMED by the DLMS AL. For details on the resulting CoAP wrapper operation see 7.3.5.7.4 and 7.3.5.7.5.

#### 7.3.6.6.3.3.2 Streaming of response APDUs over unreliable CoAP TL

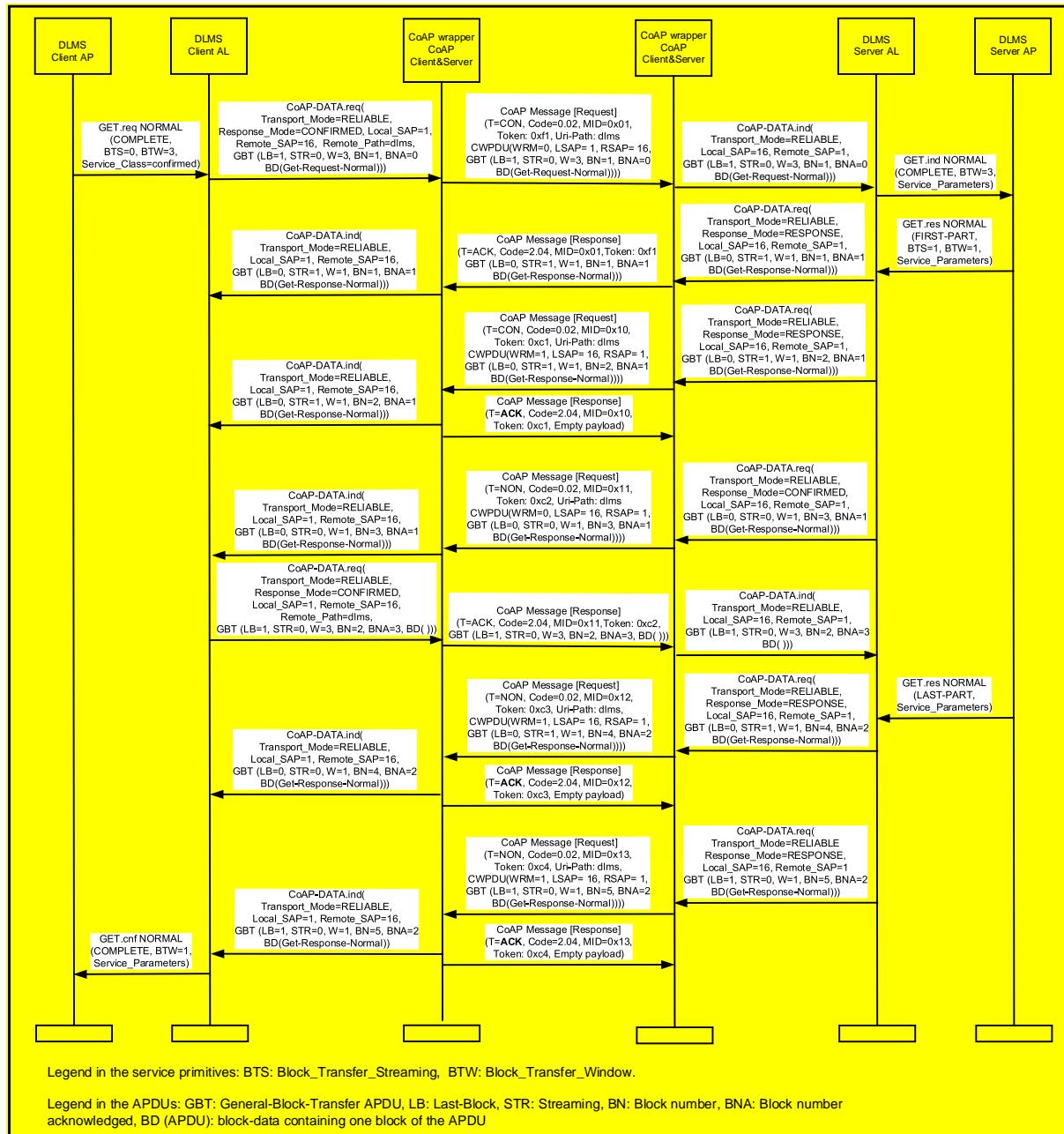
## DLMS/COSEM Architecture and Protocols



**Figure 65 – Confirmed GET service with GBT streaming over unreliable CoAP TL**

Figure 65 shows a GET service with GBT over unreliable CoAP TL with streaming of block 2,3 and 4, 5 and where the 5<sup>th</sup> block send by the DLMS Server AL is lost and recovered by GBT.

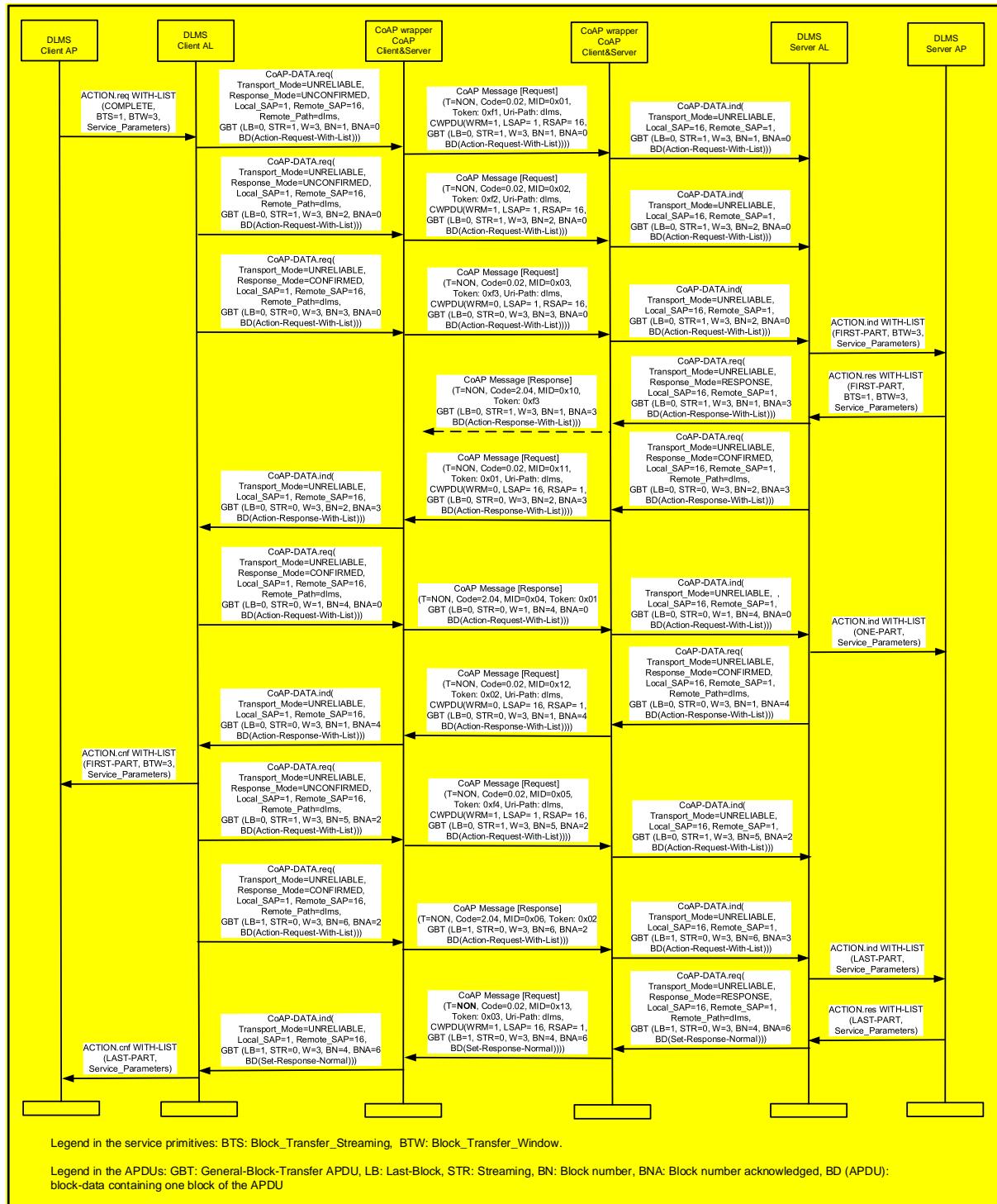
### 7.3.6.6.3.3.3 Streaming of response APDUs over reliable CoAP TL



**Figure 66 – Confirmed GET service with GBT streaming over reliable CoAP TL**

Figure 66 shows a GET service with GBT over reliable CoAP TL with streaming of block 2,3 and 4, 5.

### 7.3.6.6.3.4 GBT procedure with streaming in both directions through CoAP TL



**Figure 67 – Confirmed service request with GBT streaming in both directions over unreliable CoAP TL**

Figure 67 shows an ACTION-WITH-LIST service with bi-directional GBT and block recovery over unreliable CoAP TL. Compare Figure 124 [GB Ed. 10].

**7.3.6.6.3.5 Abort of GBT procedure through CoAP TL**

An ABORT GBT APDU is transferred as an APDU carrying an unconfirmed service invocation (i.e., by setting Response\_Mode = UNCONFIRMED in the CoAP-DATA.req primitive) by a requesting DLMS AL or as an APDU carrying a response (i.e., by set of Response\_Mode = RESPONSE in the CoAP-DATA.req primitive) by a responding DLMS AL.

## 8 Data Link Layer using the HDLC protocol

### 8.1 Overview

#### 8.1.1 General

This chapter specifies the data link layer for the 3-layer, connection-oriented, HDLC based, asynchronous communication profile.

This specification supports the following communication environments:

- point-to-point and point-to-multipoint configurations;
- dedicated and switched data transmission facilities;
- half-duplex and full-duplex connections;
- asynchronous start/stop transmission, with 1 start bit, 8 data bits, no parity, 1 stop bit.

Two special procedures are also defined:

- transferring of separately received Service User layer PDU parts from the server to the client in a transparent manner. The server side Service user layer can give its PDU to the data link layer in fragments and the data link layer can hide this fragmentation from the client, see 8.4.5.4.4 and 8.4.5.4.5;
- event reporting, by sending UI frames from the secondary station to the primary station, see 8.4.5.4.7.

Clause 4 gives an explanation of the role of data models and protocols in device data exchange.

#### 8.1.2 Structure of the data link layer

In order to ensure a coherent data link layer service specification for both connection-oriented and connectionless operation modes, the data link layer is divided into two sublayers: the Logical Link Control (LLC) sublayer and the Medium Access Control (MAC) sublayer.

The LLC sublayer is based on ISO/IEC 8802-2.

The presence of this sublayer in the connection-oriented profile is somewhat artificial: it is used as a kind of protocol selector, and the 'real' data link layer connection is ensured by the MAC sublayer. It can be considered that the standard LLC sublayer is used in an extended class I operation, where the LLC sublayer provides the standard data link connectionless services to its service user layer via a connection-oriented MAC sublayer, which executes the services.

The MAC sublayer – the major part of this data link layer specification – is based on ISO/IEC 13239. The second edition of that standard includes a number of enhancements compared to the original HDLC standard, for example in the areas of addressing, error protection, and segmentation. The third edition incorporates a new frame format, which meets the requirements of the environment found in telemetry applications for electricity metering and similar industries.

For the purpose of this Technical Report, the following selections from the HDLC standard have been made:

- unbalanced connection-mode data link operation;

**NOTE** In the DLMS/COSEM environment, the choice of an unbalanced mode of operation is natural: it is the consequence of the fact that communication in this environment is based on the client/server model.

- two-way alternate data transfer , TWA;
- the selected HDLC class of procedures is UNC – Unbalanced operation Normal response mode Class – extended with UI frames;
- frame format type 3;
- non-basic frame format transparency.

In the unbalanced connection-mode data link operation two or more stations are involved. The primary station assumes responsibility for the organization of data flow and for unrecoverable data link level error conditions, by sending command and supervisory frames. The secondary station(s) respond(s) by sending response frames.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	151/614
-----------------------	------------	-----------------------	---------

NOTE In the context of DLMS/COSEM the primary station is often, but does not have to be, the client.

The basic repertoire of commands and responses of the UNC class of procedures is extended with the UI frame to support multicasting and broadcasting and non-solicited information transfer from server to the client.

Using the unbalanced connection-mode data link operation implies that the client and server side data link layers are different in terms of the sets of HDLC frames and their state machines.

### 8.1.3 Specification method

Sublayers of the data link layer are specified in terms of **services** and **protocols**.

**Service specifications** cover the services required of, or by, the given sublayer at the logical interfaces with the neighbouring other sublayer or layer, using connection-oriented procedures. Services are the standard way to specify communications between protocol layers. Through the use of four types of transactions, commonly known as service primitives (Request, Indication, Response and Confirm) the service provider co-ordinates and manages the communication between the users. Using service primitives is an abstract, implementation-independent way to specify the transactions between protocol layers. Given this abstract nature of the primitives, their use makes good sense for the following reasons:

- they permit a common convention to be used between layers, without regard to specific operating systems and specific languages;
- they give the implementers a choice of how to implement the service primitives on a specific machine.

Service primitives include service parameters. There are three classes of service parameters:

- parameters transmitted to the peer layer, becoming part of the transmitted frame, for example addresses, control information;
- parameters, which have only local significance;
- parameters, which are transmitted transparently across the data link layer to the user of the data link.

NOTE Data link layer management services are explained in 8.6.

This Technical Report specifies values for parameters of the first category only.

As the services of the data link layer – called DL services – are in fact provided by the MAC sublayer i.e. the MA services, the two service sets are specified together in 8.2 for a concise presentation.

**Protocol specifications** for a protocol layer / sublayer include:

- the specification of the procedures for the transmission of the set of messages exchanged between peer layers;
- the procedures for the correct interpretation of protocol control information;
- the layer behaviour.

Protocol specifications for a protocol layer / sublayer do not include:

- the structure and the meaning of the information which is transmitted by means of the layer (Information field, User data subfield);
- the identity of the Service User layer;
- the manner in which the Service User layer operation is accomplished as a result of exchanging Data Link messages;
- the interactions that are the result of using the protocol layer.

The protocol for the LLC sublayer is specified in 8.3 and the protocol for the MAC sublayer is specified in 8.4.

As the MAC sublayer behaviour is quite complex, some aspects of the service invocation handling are discussed in the service specification part, although these are normally part of the protocol specification.

152/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

## 8.2 Service specification

### 8.2.1 General

This clause specifies the services required of the data link layer by the service user layer, using connection-oriented procedures.

All DL services are, in fact, provided by the MAC sublayer: the LLC sublayer transparently transmits the DL-CONNECT.xxx service primitives to/from the “real” service provider MAC sublayer as the appropriate MA-CONNECT.xxx service primitive.

As the client and the server side LLC and MAC sublayers are different, service primitives are specified for both sides.

The addressing scheme for the MAC sublayer is specified in 8.4.2.

### 8.2.2 Setting up the data link connection: the DL-CONNECT and MA-CONNECT services

#### 8.2.2.1 Overview

Figure 68 shows the services required of the primary station (client side) and secondary station (server side) data link layers by the service user layer for data link connection establishment.

Data link connection establishment can be requested by the client side only. Consequently, the DL-CONNECT / MA-CONNECT .request and .confirm primitives are provided only at the client (primary station) side. On the other hand, the MA-CONNECT / DL-CONNECT .indication and .response primitives are provided only at the server (secondary station) side.

The DL-CONNECT / MA-CONNECT .request primitives – in case of a locally detected error – can be also locally confirmed.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	153/614
-----------------------	------------	-----------------------	---------

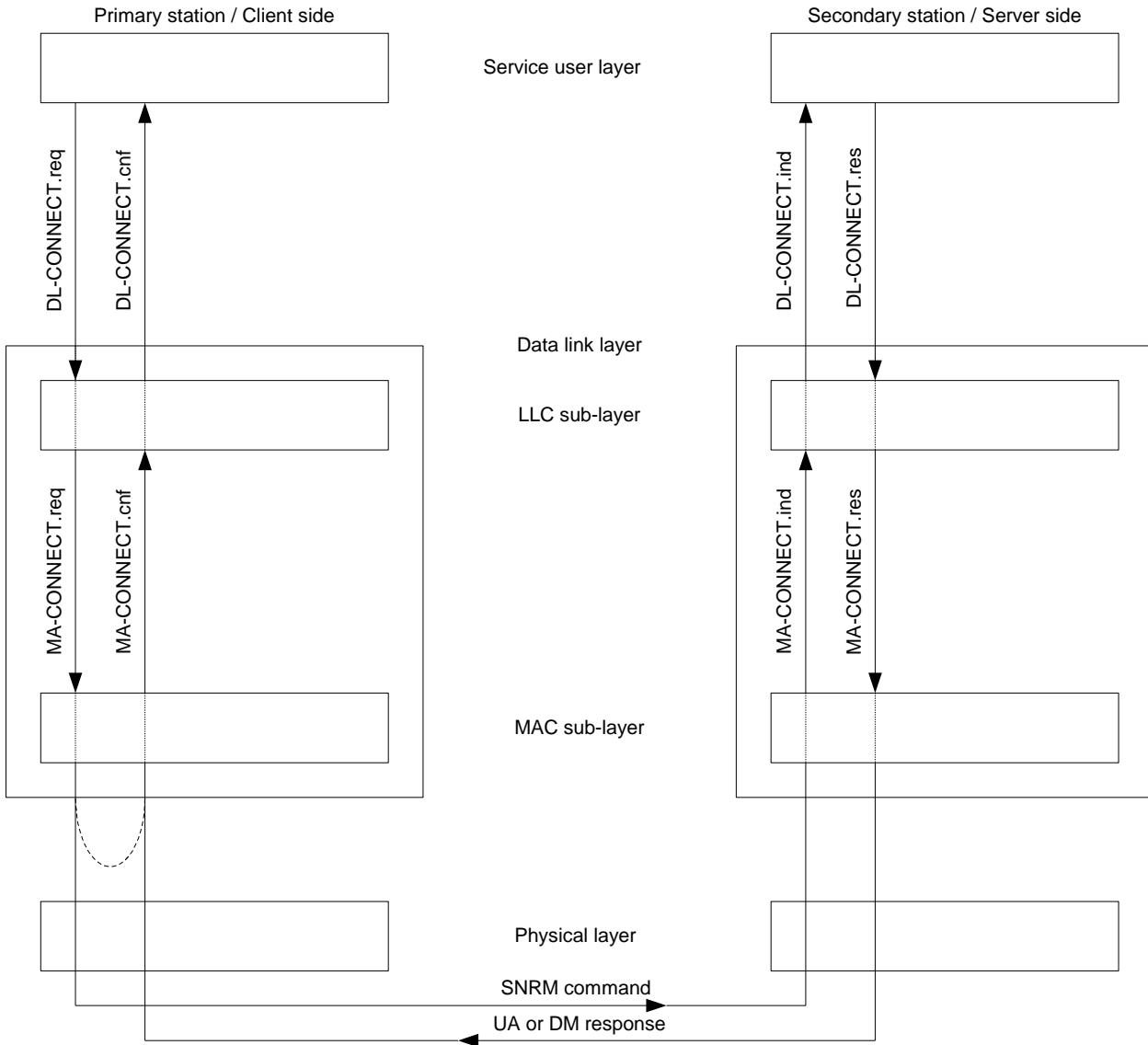


Figure 68 – Data link layer services for data link connection

### 8.2.2.2 DL-CONNECT.request and MA-CONNECT.request

#### 8.2.2.2.1 Function

These primitives are the service request primitives for the connection establishment service.

#### 8.2.2.2.2 Semantics of the service primitives

The primitives shall provide parameters as follows:

<b>DL-CONNECT.request</b> ( Destination_MSAP <sup>5</sup> , Source_MSAP, User_Information )	<b>MA-CONNECT.request</b> ( Destination_MSAP, Source_MSAP, User_Information )
--	--

<sup>5</sup> MSAP in this environment is equal to the HDLC address.

Where:

- Destination\_MSAP and Source\_MSAP identify the referenced data link layer connection to be set up.
- User\_Information is optional. The specification of its contents is not within the Scope of this Technical Report.

#### **8.2.2.2.3 Use**

The DL-CONNECT.request primitive is invoked by the service user layer to request the set-up of a data link connection. The MA-CONNECT.request primitive is invoked then by the LLC-sublayer. Upon reception of this primitive, the MAC sublayer sends out a correctly formatted SNRM frame. If present, the User\_Information parameter is inserted in the User data subfield of the Information field of the SNRM frame.

NOTE SNRM is the HDLC frame used for requesting an MAC connection establishment, see 8.4.3.5.

#### **8.2.2.3 DL-CONNECT.indication and MA-CONNECT.indication**

##### **8.2.2.3.1 Function**

These primitives are the service indication primitives for the connection establishment service.

##### **8.2.2.3.2 Semantics of the service primitives**

The primitives shall provide parameters as follows:

<b>DL-CONNECT.indication (</b> Destination_MSAP, Source_MSAP, User_Information <b>)</b>	<b>MA-CONNECT.indication (</b> Destination_MSAP, Source_MSAP, User_Information <b>)</b>
---	---

Where:

- Destination\_MSAP and Source\_MSAP identify the referenced data link layer connection to be set up.
- User\_Information parameter is optional. The specification of its contents is not within the Scope of this Technical Report.

#### **8.2.2.3.3 Use**

The MA-CONNECT.indication primitive is generated by the MAC sublayer, following the reception of a correctly formatted SNRM frame, to indicate that the peer MAC sublayer requested the establishment of an MAC Connection. If present, the contents of the User data subfield of the Information field of the SNRM frame is passed to the service user layer as the User\_Information parameter.

NOTE If the secondary station receives an SNRM frame with the address parameters of an already existing data link connection, it responds with an UA frame and sets the receive and send state variables to zero. See also 8.4.3.5.

The DL-CONNECT.indication primitive is generated then by the LLC sublayer to indicate to the service user layer that the peer data link layer requested a Data Link connection.

#### **8.2.2.4 DL-CONNECT.response and MA-CONNECT.response**

##### **8.2.2.4.1 Function**

These primitives are the service response primitives for the connection establishment service.

##### **8.2.2.4.2 Semantics of the service primitives**

The primitives shall provide parameters as follows:

<b>DL-CONNECT.response</b> (	<b>MA-CONNECT.response</b> (
Destination_MSAP,	Destination_MSAP,
Source_MSAP,	Source_MSAP,
Result,	Result,
User_Information	User_Information
)	)

Where:

- Destination\_MSAP and Source\_MSAP identify the referenced data link layer connection being set up.
- Result indicates whether the proposed connection could be accepted or not, and whether a response frame should be sent or not. It can have one of the following values:
  - Result == OK. This means that the received connect request can be accepted by the service user layer;
  - Result == NOK. This means that the received connect request cannot be accepted by the service user layer;
  - Result == NO-RESPONSE. This means that no response to the DL-CONNECT.indication shall be sent.

NOTE The Result parameter indicates only whether the data link connection can or cannot be accepted by the service user higher layers. The data link layer itself may refuse a proposed connection, (for example because it supports only one connection at a given moment, thus it is not able to support a second one) even if the higher layers could accept it (Result == OK).

User\_Information is optional. It may be present only when Result == NOK. The specification of its contents is not within the Scope of this Technical Report.

#### 8.2.2.4.3 Use

The DL-CONNECT.response primitive is invoked by the service user layer to indicate to the data link layer whether the data link layer connection requested previously can be accepted or not.

The DL-CONNECT.response primitive with Result == OK is invoked by the service user layer to indicate to the data link layer that the proposed data link layer connection has been accepted. The LLC sublayer invokes then the MA-DISCONNECT.response primitive. The MAC layer sends out an UA frame.

The DL-CONNECT.response primitive with Result == NOK is invoked by the service user layer to indicate to the data link layer that the proposed data link layer connection has not been accepted. The LLC sublayer invokes then the MA-CONNECT.response primitive. The MAC sublayer sends a DM frame to its peer. In this case the User\_Information parameter, if present, is inserted in the User data subfield of the Information field of this DM frame.

The DL-CONNECT.response primitive with Result == NO-RESPONSE is invoked by the service user layer when no response should be sent. The LLC sublayer invokes then the MA-CONNECT.response primitive. The MAC layer does not send any frame.

#### 8.2.2.5 DL-CONNECT.confirm and MA-CONNECT.confirm

##### 8.2.2.5.1 Function

These primitives are the service .confirm primitives for the connection establishment service.

### 8.2.2.5.2 Semantics of the service primitives

The primitives shall provide parameters as follows:

<b>DL-CONNECT.confirm</b>	(	<b>MA-CONNECT.confirm</b>	(
Destination_MSAP,		Destination_MSAP,	
Source_MSAP,		Source_MSAP,	
Result,		Result,	
User_Information	)	User_Information	)

Where:

- Destination\_MSAP and Source\_MSAP reference data link layer connection, which is confirmed by the service primitive.
- Result indicates the result of the DL-CONNECT / MA-CONNECT.request service invocation invoked previously. It can have one of the following values:
  - Result == OK. This means that the connect request was accepted by the remote station;
  - Result == NOK-REMOTE. This means that the connect request was not accepted by the remote station;
  - Result == NOK-LOCAL. This means that a local error has occurred, for example the service user layer tried to establish an already existing data link connection;
  - Result == NO-RESPONSE. This means that there was no response from the remote station to the connect request.

User\_Information is optional. It may be present only when the Result is NOK-REMOTE. The specification of its contents is not within the Scope of this Technical Report.

### 8.2.2.5.3 Use

The MA-CONNECT.confirm primitive is generated by the MAC sublayer to indicate to the LLC sublayer the result of a MA-CONNECT.request service invocation received previously. The DL-CONNECT.confirm primitive is generated then by the LLC sublayer to indicate to the service user layer the result of a DL-CONNECT.request service invocation received previously.

If the received frame is a DM frame, the contents of the User data subfield of the Information field is passed to the service user layer as the User\_Information parameter.

## 8.2.3 Disconnecting the data link connection: the DL-DISCONNECT and MA-DISCONNECT services

### 8.2.3.1 Overview

Figure 69 shows the services required of the primary station (client side) and secondary station (server side) data link layers by the service user layer for data link connection termination.

Data link disconnection termination can be requested by the client side only. Consequently, the DL-DISCONNECT / MA-DISCONNECT.request and .confirm primitives are provided only at the client (primary station) side. On the other hand, the remotely initiated (by the client) DL-DISCONNECT / MA-DISCONNECT.indication and .response service primitives are provided only at the server (secondary station) side.

NOTE When this data link layer is used together with the DLMS/COSEM AL as defined in Clause 9, DL-DISCONNECT services are used to release existing AAs.

Both the client and server side LLC and MAC sublayers provide a locally generated DL-DISCONNECT / MA-DISCONNECT.indication primitive, to signal a non-solicited disconnection, due to an unexpected loss of the data link and/or physical connection. A MAC connection loss is locally detected at the MAC layer. A physical connection loss is indicated locally by the PH-ABORT.indication primitive.

The DL-DISCONNECT / MA-DISCONNECT.request primitives – in the case of a locally detected error – can be also locally confirmed.

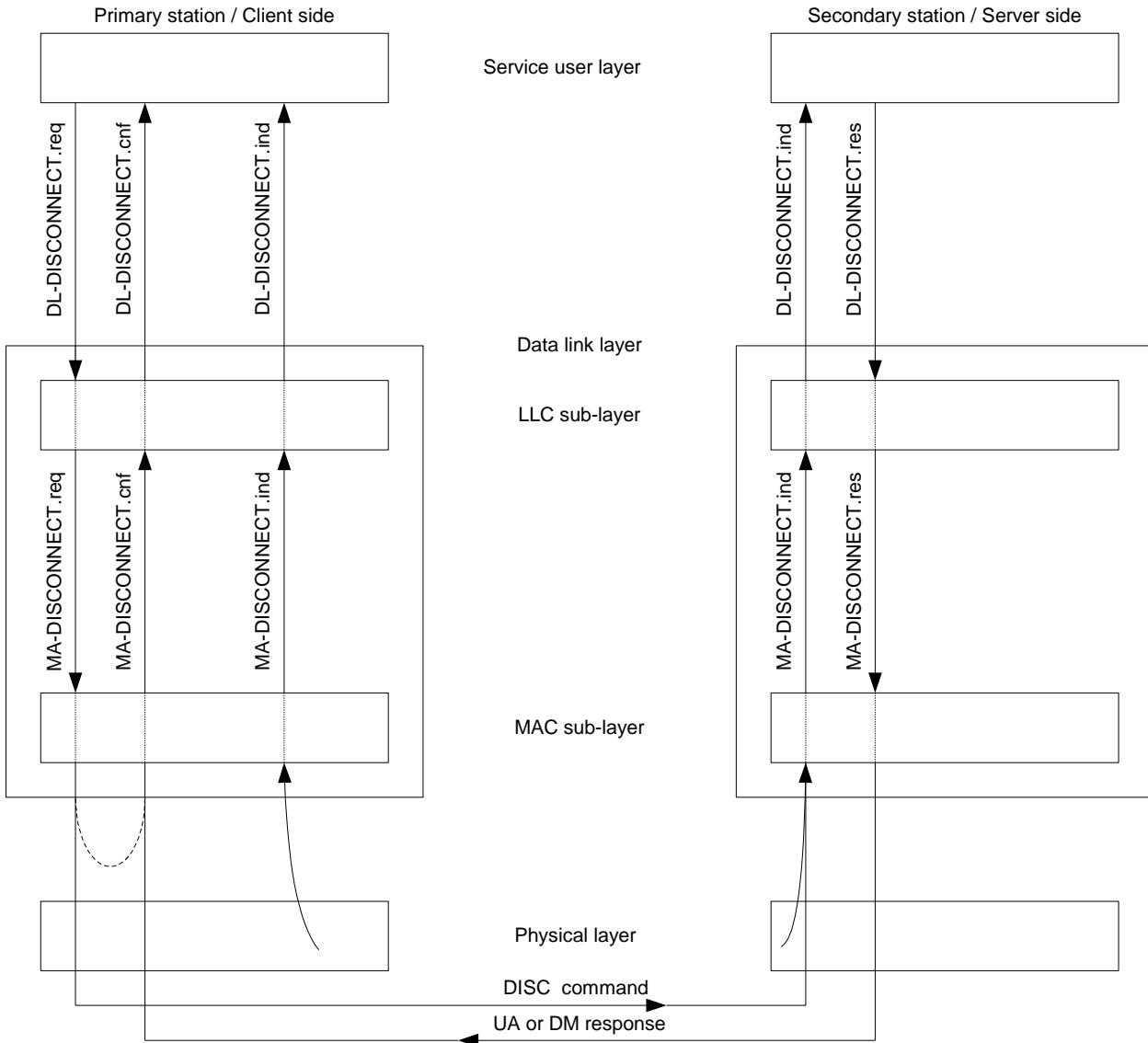


Figure 69 – Data link layer services for data link disconnection

### 8.2.3.2 DL-DISCONNECT.request and MA-DISCONNECT.request

#### 8.2.3.2.1 Function

These primitives are the service request primitives for the connection termination service.

#### 8.2.3.2.2 Semantics of the service primitives

The primitives shall provide parameters as follows:

**DL-DISCONNECT.request (**

Destination\_MSAP,  
Source\_MSAP,  
User\_Information  
)

**MA-DISCONNECT.request (**

Destination\_MSAP,  
Source\_MSAP,  
User\_Information  
)

Where:

- Destination\_MSAP and Source\_MSAP identify the data link layer connection to be terminated.
- User\_Information is optional. The specification of its contents is not within the Scope of this Technical Report.

### 8.2.3.2.3 Use

The DL-DISCONNECT.request primitive is invoked by the service user layer to request the termination of an existing data link layer connection. The MA-DISCONNECT.request primitive is invoked then by the LLC sublayer. Upon reception of this primitive, the MAC sublayer sends out a correctly formatted DISC frame. If present, the User\_Information parameter is inserted in the User data subfield of the Information field of this DISC frame.

### 8.2.3.3 DL-DISCONNECT.indication and MA-DISCONNECT.indication

#### 8.2.3.3.1 Function

These primitives are the service indication primitives for the connection termination service.

#### 8.2.3.3.2 Semantics of the service primitive

The primitives shall provide parameters as follows:

DL-DISCONNECT.indication (	MA-DISCONNECT.indication (
Destination_MSAP,	Destination_MSAP,
Source_MSAP,	Source_MSAP,
Reason,	Reason,
Unnumbered_Send_Status,	Unnumbered_Send_Status,
User_Information	User_Information
)	)

Where:

- Destination\_MSAP and Source\_MSAP identify the data link layer connection to be terminated.
- Reason indicates the reason for the DL-DISCONNECT.indication invocation. It can have one of the following values:
  - Reason == REMOTE: the data link layer received a disconnection request from the client side. This case may happen only at the server side;
  - Reason == LOCAL-DL: there was a fatal data link connection failure;
  - Reason == LOCAL-PHY: there was a fatal physical connection failure. These two latter cases may occur both on the client and on the server side.
- The value of the Unnumbered\_Send\_Status, USS, parameter indicates whether at the moment of the DL-DISCONNECT / MA-DISCONNECT .indication primitive invocation the data link layer resp. the MAC sublayer has (USS == TRUE) or does not have (USS == FALSE) pending UI message(s).

NOTE 1 The number of pending UI frames is a layer parameter, which can be accessed using the layer management services.

- User\_Information is optional. It may be present only when Reason == REMOTE. If present, it carries the contents of the User data subfield of the Information field of the DISC frame received. The specification of its contents is not within the Scope of this Technical Report.

#### 8.2.3.3.3 Use

The MA-DISCONNECT.indication primitive with Reason == REMOTE is generated by the server MAC sublayer when it receives a correctly formatted DISC frame. If present, the contents of the User data subfield of the Information field of the DISC frame is passed to the service user layer as the User\_Information parameter.

The MA-DISCONNECT.indication primitive with Reason == LOCAL-DL is generated by the MAC sublayer to indicate that a fatal failure occurred in the MAC sublayer.

The MA-DISCONNECT.indication primitive with Reason == LOCAL-PHY is generated by the MAC sublayer after receiving a Ph-ABORT.indication service primitive from the PhL, meaning that the physical connection has been interrupted.

The DL-DISCONNECT.indication primitive is generated then by LLC sublayer to indicate to the service layer that the disconnection of the data link layer has been requested.

NOTE 2 The Service User layer cannot refuse this request, but it may indicate that no response should be sent.

### 8.2.3.4 DL-DISCONNECT.response and MA-DISCONNECT.response

#### 8.2.3.4.1 Function

These primitives are the service response primitives for the connection termination service.

#### 8.2.3.4.2 Semantics of the service primitives

The primitives shall provide parameters as follows:

<b>DL-DISCONNECT.response</b>	<b>(</b>	<b>MA-DISCONNECT.response</b>	<b>(</b>
Destination_MSAP,		Destination_MSAP,	
Source_MSAP,		Source_MSAP,	
Result		Result	
<b>)</b>		<b>)</b>	

Where:

- Destination\_MSAP and Source\_MSAP identify the data link layer connection being terminated.
- Result can have one of the following values:
  - Result == OK: the disconnect request received refers to an existing higher layer connection;
  - Result == NOK: the disconnect request received refers to a non-existing higher layer connection;
  - Result == NO-RESPONSE: no response to the DL-DISCONNECT.indication shall be sent.

#### 8.2.3.4.3 Use

The DL-DISCONNECT.response primitive is invoked by the service user layer to indicate to the data link layer whether the data link disconnection requested previously can be accepted or not. As in this environment the server has no right to refuse the disconnection, the response depends only on the existence of the referenced Data Link connection.

The DL-DISCONNECT.response primitive with RESULT == OK is invoked by the service user layer when the disconnection request refers to an existing higher layer connection. The LLC sublayer invokes then the MA-DISCONNECT.response primitive. The MAC sublayer enters into the disconnected state and it sends a UA message to its peer sublayer.

The DL-DISCONNECT.response primitive with RESULT == NOK is invoked by the service user layer when the disconnection request refers to a non-existing higher layer connection. The LLC sublayer invokes then the MA-DISCONNECT.response primitive. The MAC sublayer, depending on its connection state, enters into (remain in) a disconnected state and it sends a UA or a DM frame to its peer as appropriate.

NOTE UA and DM are the appropriate HDLC frames to be sent following the invocation of the DL-DISCONNECT.res (MA-DISCONNECT.res) service primitive.

The DL-DISCONNECT.response primitive with RESULT == NO-RESPONSE is invoked by the service user layer when no response shall be sent. The LLC sublayer invokes then the MA-DISCONNECT.response primitive. The MAC layer does not send any frame.

### 8.2.3.5 DL-DISCONNECT.confirm and MA-DISCONNECT.confirm

#### 8.2.3.5.1 Function

These primitives are the service .confirm primitives for the connection termination service.

#### 8.2.3.5.2 Semantics of the service primitives

The primitives shall provide parameters as follows:

<b>DL-DISCONNECT.confirm</b>	<b>(</b>	<b>MA-DISCONNECT.confirm</b>	<b>(</b>
Destination_MSAP,		Destination_MSAP,	
Source_MSAP,		Source_MSAP,	
Result		Result	
<b>)</b>		<b>)</b>	

Where:

- Destination\_MSAP and Source\_MSAP identify the data link layer connection, the termination of which is confirmed by the service primitive.
- Result can have one of the following values:
  - Result == OK: the disconnect request was accepted by the remote station;
  - Result == NOK: the disconnect request was not accepted by the remote station;
  - Result == NO-RESPONSE: there was no response from the remote station to the disconnect request.

#### 8.2.3.5.3 Use

The MA-DISCONNECT.confirm primitive is generated by the MAC sublayer in order to indicate to LLC sublayer the result of a MA-DISCONNECT.request service invocation received previously. The DL-DISCONNECT.confirm primitive is generated then by LLC sublayer to indicate to the service user layer the result of a DL-DISCONNECT.request service invocation received previously. These service primitives can be generated remotely or locally.

### 8.2.4 Data transfer: the DL-DATA and MA-DATA services

#### 8.2.4.1 Overview

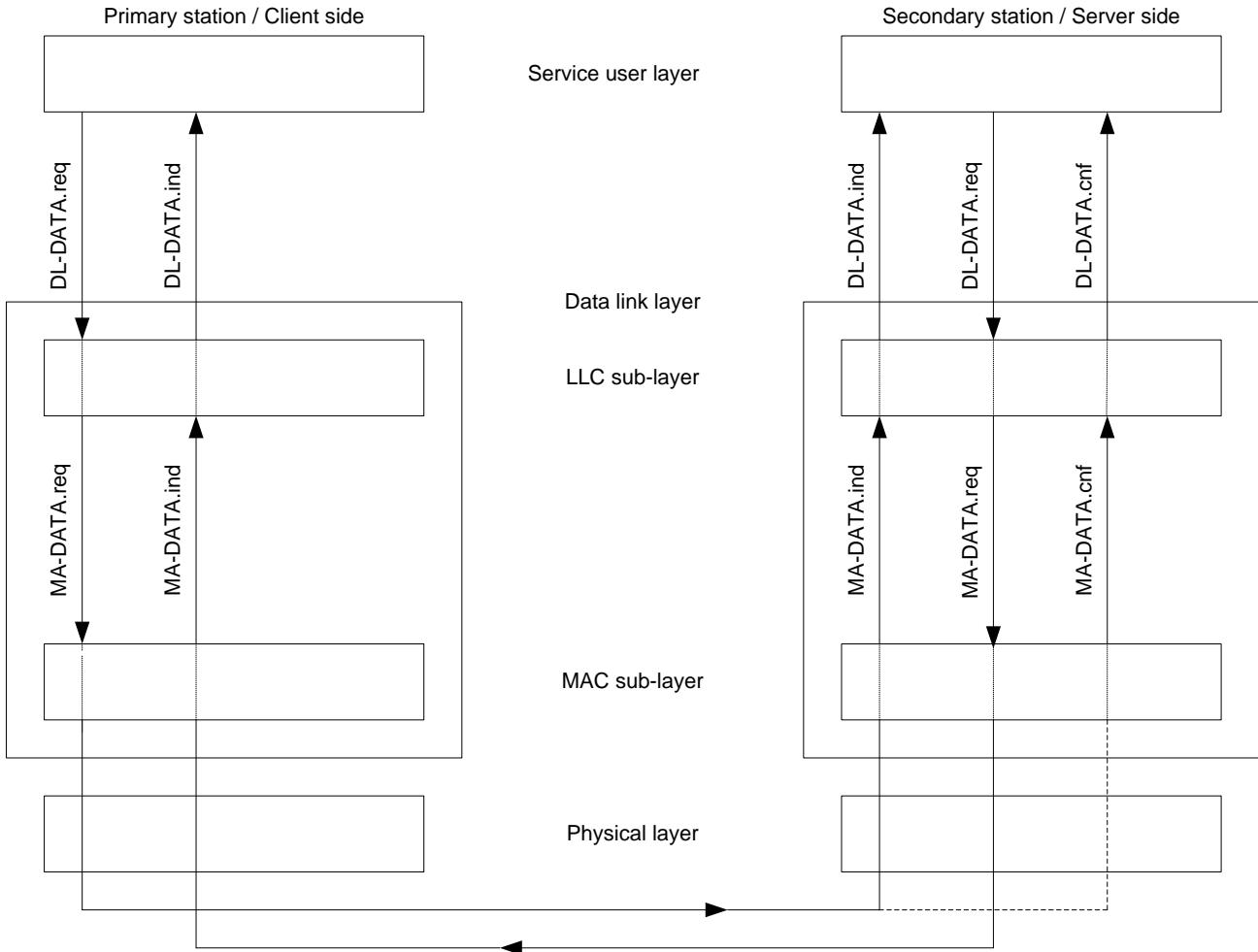
Figure 70 shows the services required of the primary station (client side) and secondary station (server side) data link layers by the service user layer for data transfer, using I frames or UI frames of the MAC sublayer.

The client and the server sides provide basically the same service set. However, in addition to the two standard .request and .indication primitives, a DL-DATA / MA-DATA .confirm primitive is also provided at the server side. This is necessary for transparent long message transfer. See also 8.4.5.4.5.

The client and server MAC sublayers behave differently with regard to the DL-DATA / MA-DATA .request primitive: In the Normal Response Mode used, the server station may initiate transmission only as the result of receiving an explicit permission to do so from the client station.

After giving the permission to talk to the server, the client has also to wait for the server (with a specific Time-out, TO\_WAIT\_RESP, see 8.4.5.6.1) to give back explicitly this permission before initiating a new transmission. If no response is received however, the client re-gains the permission to talk when this time-out expires.

Procedures for data exchange for the LLC layer are specified in 8.3 and for the MAC sublayer in 8.4.5.4.

**Figure 70 – Data link layer data transfer services****8.2.4.2 DL-DATA.request and MA-DATA.request****8.2.4.2.1 Function**

These primitives are the service request primitives for the data transfer service.

**8.2.4.2.2 Semantics of the service primitives**

The primitives shall provide parameters as follows:

<b>DL-DATA.request</b> <code>(     Destination_LSAP,     Source_LSAP,     LLC_Quality,     Destination_MSAP,     Source_MSAP,     Frame_type,     Data )</code>	<b>MA-DATA.request</b> <code>(     Destination_MSAP,     Source_MSAP,     Frame_type,     Data )</code>
--	--

Where:

- Destination\_LSAP and Source\_LSAP identify the data link layer connection referenced. The value of the LLC\_Quality parameter is used as the Control field of the PDU. See 8.3.2.
- Destination\_MSAP and Source\_MSAP identify the remote and local MSAPs involved in the data unit transmission. The Destination\_MSAP can be an individual address, a group address, or a special HDLC address (All-station, No-station, etc.). Please refer to 8.4.2.4.

- Frame\_Type indicates for the MAC sublayer which type of HDLC frame shall be sent. Valid frame types are different for the client and server sides:
  - On the client side, valid frame types are I-COMPLETE and UI;
  - On the server side valid frame types are I-COMPLETE, I-FIRST-FRAGMENT, I-FRAGMENT, I-LAST-FRAGMENT, and UI. See also 8.4.3.
- Data contains the service user protocol data unit (LSDU resp. MSDU) to be transferred to the peer layer. The MSDU parameter may be empty (for example when Frame\_type == UI, but the UI frame contains an empty Information Field).

#### 8.2.4.2.3 Use

The DL-DATA.request primitive is invoked by the service user layer entity to request sending of a protocol data unit to a single peer application entity or, in the case of multicasting and broadcasting, to multiple peer application entities.

The receipt of this primitive shall cause the LLC sublayer to append the LLC specific fields (the two LLC addresses and the LLC\_Quality parameter) to the received LSDU, and to pass the properly formed LPDU to the MAC sublayer (by invoking the MA-DATA.request primitive) for transferring it to the peer LLC sublayer. The MAC sublayer sends then the appropriate frame type.

NOTE When Frame\_type == I-FRAGMENT or I-LAST-FRAGMENT, no LLC specific fields are appended to the LSDU.

#### 8.2.4.3 DL-DATA.indication and MA-DATA.indication

##### 8.2.4.3.1 Function

These primitives are the service indication primitives for the data transfer service.

##### 8.2.4.3.2 Semantics of the service primitives

The primitive shall provide parameters as follows:

<b>DL-DATA.indication</b>	(	<b>MA-DATA.indication</b>	(
Destination_LSAP,		Destination_MSAP,	
Source_LSAP,		Source_MSAP,	
LLC_Quality,		Frame_type,	
Destination_MSAP,		Data	
Source_MSAP,			)
Frame_type,			
Data	)		

Where:

- Destination\_LSAP and Source\_LSAP identify the data link layer connection referenced. The value of the LLC\_Quality parameter is used as the Control field of the PDU. See 8.3.2.
- Destination\_MSAP and Source\_MSAP identify the local and remote MSAPs involved in the data unit transmission. The Destination\_MSAP can be an individual address, a group address, or a special HDLC address (All-station, No-station, etc.). Please refer to 8.4.2.4.
- Frame\_Type indicates to the Service User layer the type of the HDLC frame received. Valid frame types are at both the client and the server sides: I-COMPLETE and UI. An I-COMPLETE frame shall be reported only if it has been received within an established MAC connection.
- Data contains the service user layer protocol data unit (LSDU resp. MSDU) received from the peer layer. The MSDU parameter may be empty (for example when Frame\_Type == UI, but the UI frame contains an empty Information Field).

##### 8.2.4.3.3 Use

The MA-DATA.indication primitive is passed from the MAC sublayer entity to the LLC sublayer to indicate the arrival of an MPDU from a remote MAC entity to the local MAC entity.

The LLC sublayer checks then the LLC addresses. If correct, it removes the LLC specific fields – the two LLC addresses and the LLC\_Quality parameter – from the received LPDU, and passes the remaining, properly formatted LSDU to the service user protocol layer with the help of the DL-DATA.indication service primitive. Otherwise it discards the LPDU received.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	163/614
-----------------------	------------	-----------------------	---------

#### 8.2.4.4 DL-DATA.confirm and MA-DATA.confirm

##### 8.2.4.4.1 Function

These primitives are the service .confirm primitives for the data transfer service.

##### 8.2.4.4.2 Semantics of the service primitives

The primitives shall provide parameters as follows:

<b>DL-DATA.confirm</b>	<b>(</b>	<b>MA-DATA.confirm</b>	<b>(</b>
Destination_LSAP,		Destination_MSAP,	
Source_LSAP,		Source_MSAP,	
LLC_Quality,		Frame_type,	
Destination_MSAP,		Result	
Source_MSAP,		)	
Frame_type,			
Result			
)			

Where:

- Destination\_LSAP and Source\_LSAP identify the data link layer connection referenced.
- Destination\_MSAP and Source\_MSAP identify the local and remote MSAP-s involved in the data unit transmission. The Destination\_MSAP shall be an individual address.
- Frame\_Type indicates the type of the HDLC frame which is confirmed. Valid frame types are I-FIRST-FRAGMENT, I-FRAGMENT and I-LAST-FRAGMENT.
- Result indicates the result of the transmission of the received MSDU resp. LSDU. Possible values are OK and NOK.

##### 8.2.4.4.3 Use

The MA-DATA.confirm primitive is generated by MAC sublayer in order to indicate to the service user layer the result of a previously received MA-DATA.request service, when this .request service has been invoked with Frame\_type = I-FIRST-FRAGMENT, I-FRAGMENT or I-LAST-FRAGMENT. These frame types are used with a special procedure specified for transferring long messages from the server to the client, as specified in 8.4.5.4.5. The primitive is generated when the MSDU requested previously has been successfully sent (following the receipt of the positive acknowledgement after sending out the last HDLC frame) to the peer MAC sublayer.

The DL-DATA.confirm primitive is generated then by LLC sublayer to the service user layer.

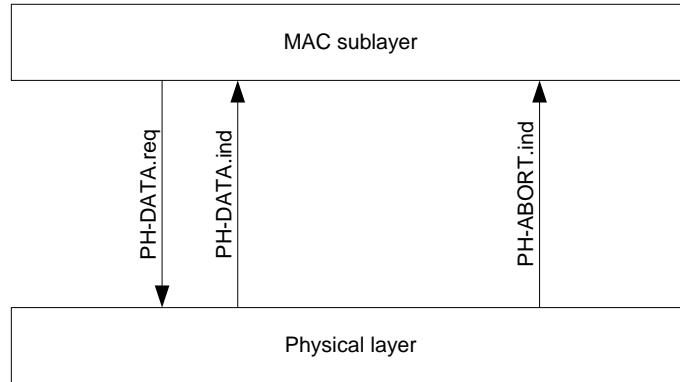
#### 8.2.5 Physical layer services used by the MAC sublayer

##### 8.2.5.1 Overview

Figure 71 shows services provided by the PhL to the MAC sublayer. The same service set is used both at the client and the server sides.

##### 8.2.5.2 Setting up a physical link

Setting up the physical link does not utilize services of protocol layers above the PhL.

**Figure 71 – Physical layer services used by the MAC sublayer**

#### 8.2.5.3 Disconnecting the physical link

Disconnecting the physical link does not utilize services of protocol layers above the PhL. The only service used by another protocol layer is the PH-ABORT.indication service (see Clause 5), to generate an MA-DISCONNECT.indication service primitive with Reason == LOCAL\_PH.

#### 8.2.5.4 Data transfer

The MAC sublayer uses the PH-DATA.request and .indication service primitives of the PhL for exchanging data with a remote device.

### 8.3 Protocol specification for the LLC sublayer

#### 8.3.1 Role of the LLC sublayer

The LLC sublayer transmits LSDUs transparently between its service user layer and the MAC sublayer.

#### 8.3.2 LLC PDU format

The standard LLC PDU format is shown in Figure 72:

Destination (remote) LSAP	Source (local) LSAP	Control	Information
8 bits	8 bits	8 or 16 bits	n*8 bits

**Figure 72 – The ISO/IEC 8802-2 LLC PDU format**

For the purposes of DLMS/COSEM, this LLC PDU format is used as shown on Figure 73:

Destination (remote) LSAP	Source (local) LSAP	LLC_Quality	Information
8 bits: 0xE6	8 bits: 0xE6 or 0xE7	8 bits: 0x00	n*8 bits

**Figure 73 – LLC format as used in DLMS/COSEM**

- The value of the Destination\_LSAP is 0xE6;
- The value of the Source\_LSAP is 0xE6 or 0xE7. The least significant bit is used as a command/response identifier. When set to 0, it identifies a ‘command’ and when set to 1 it identifies a “response”;
- The Control byte is referred here to as the LLC\_Quality parameter. It is reserved for future use. Its value is administered by the DLMS UA. Currently, it must be set always to 0x00;
- The information field consists of an integral number (including zero) of octets and it carries the LSDU.

The destination LSAP 0xFF is used for broadcasting purposes. Devices in this environment shall never send messages with this broadcast address, but they shall accept messages containing this broadcast destination address as if it would be addressed to them.

### 8.3.3 State transition tables for the LLC sublayer

As the role of the LLC sublayer is limited to protocol selection, its state transition diagram is quite simple: after being initialized, the LLC sublayer enters into its only stable state, the IDLE state, and returns into this state after any possible events. The state transitions of the client side and server side LLC sublayers are shown in Table 11 and in Table 12 respectively.

**Table 11 – State transition table of the client side LLC sublayer**

Current state	Event	Action	Next state
IDLE	Receive DL-CONNECT.request	Invoke MA-CONNECT.request	IDLE
IDLE	Receive MA-CONNECT.confirm	Generate DL-CONNECT.confirm	IDLE
IDLE	Receive DL-DISCONNECT.request	Invoke MA-DISCONNECT.request	IDLE
IDLE	Receive MA-DISCONNECT.indication	Generate DL-DISCONNECT.indication	IDLE
IDLE	Receive MA-DISCONNECT.confirm	Generate DL-DISCONNECT.confirm	IDLE
IDLE	Receive DL-DATA.request	Add LLC addresses and control byte (3 bytes) to the received LSDU; Invoke MA-DATA.request; see footnote 6)	IDLE
IDLE	Receive MA-DATA.indication	Check LLC addresses (3 bytes); see footnote 6) If address == OK { remove LLC addresses; generate DL-DATA.indication; } else { discard the received packet; }	IDLE

**Table 12 – State transition table of the server side LLC sublayer**

Current state	Event	Action	Next state
IDLE	Receive MA-CONNECT.indication	Generate DL-CONNECT.indication	IDLE
IDLE	Receive DL-CONNECT.response	Invoke MA-CONNECT.response	IDLE
IDLE	Receive MA-DISCONNECT.indication	Generate DL-DISCONNECT.indication	IDLE
IDLE	Receive DL-DISCONNECT.response	Invoke MA-DISCONNECT.response	IDLE
IDLE	Receive DL-DATA.request and Frame_type is I-COMPLETE, UI or I-FIRST-FRAGMENT	Add LLC addresses and control byte to the received LSDU; Invoke MA-DATA.request;	IDLE
IDLE	Receive DL-DATA.request and Frame_type is I-FRAGMENT or I-LAST-FRAGMENT	Invoke MA-DATA.request	IDLE
IDLE	Receive MA-DATA.indication	Check LLC addresses; If address == OK { remove LLC addresses; generate DL-DATA.indication; } else { discard the received packet; }	IDLE
IDLE	Receive MA-DATA.confirm	Generate DL-DATA.confirm	IDLE

<sup>6</sup> LLC specific fields are not present, when Frame\_type == I-FRAGMENT or I-LAST-FRAGMENT. See also the NOTE to 8.2.4.2.

## 8.4 Protocol specification for the MAC sublayer

### 8.4.1 The MAC PDU and the HDLC frame

#### 8.4.1.1 HDLC frame format type 3

The MAC sublayer uses the HDLC frame format type 3 as defined in Annex H.4 of ISO/IEC 13239. It is shown on Figure 74:

Flag	Frame format	Dest. address	Src. address	Control	HCS	Information	FCS	Flag
------	--------------	---------------	--------------	---------	-----	-------------	-----	------

**Figure 74 – MAC sublayer frame format (HDLC frame format type 3)**

This frame format is used in those environments where additional error protection, identification of both the source and the destination, and/or longer frame sizes are needed. Type 3 requires the use of the segmentation subfield, thus reducing the length field to 11 bits. Frames that do not have an information field, for example as with some supervisory frames, or an information field of zero length do not contain an HCS and an FCS, only an FCS. The HCS and FCS polynomials will be the same. The HCS shall be 2 octets in length.

The elements of the frame are described in the following clauses.

#### 8.4.1.2 Flag field

The length of the flag field is one byte and its value is 0x7E. When two or more frames are transmitted continuously, a single flag is used as both the closing flag of one frame and the opening flag of the next frame, as it is shown in Figure 75.

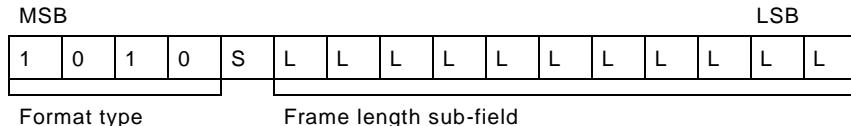
**NOTE** Frames are transmitted continuously when the period of time between two transmitted characters does not exceed the specified max. inter-octet time. See 8.4.4.3.3.

Flag	Frame I	Flag	Frame I+1	Flag	Frame I+2	Flag
------	---------	------	-----------	------	-----------	------

**Figure 75 – Multiple frames**

#### 8.4.1.3 Frame format field

The length of the frame format field is two bytes. It consists of three sub-fields referred to as the Format type sub-field (4 bit), the Segmentation bit (S, 1 bit) and the frame length sub-field (11 bit), as it is shown in Figure 76:



**Figure 76 – The frame format field**

The value of the format type sub-field is 1010 (binary), which identifies a frame format type 3 as defined in 8.4.1.1.

Rules of using the segmentation bit are defined in 8.4.5.4.4.

The value of the frame length subfield is the count of octets in the frame excluding the opening and closing frame flag sequences.

#### 8.4.1.4 Destination and source address fields

There are exactly two address fields in this frame: a destination and a source address field.

#### 8.4.1.5 Control field

The length of the control field is one byte. It indicates the type of commands or responses, and contains sequence numbers, where appropriate (frames I, RR and RNR). See also 8.4.3.

#### 8.4.1.6 Header check sequence (HCS) field

The length of the HCS field is two bytes. This check sequence is applied to only the header, i.e., the bits between the opening flag sequence and the header check sequence. Frames that do not have an information field or have an empty information field, e.g., as with some supervisory frames, do not contain an HCS and FCS, only an FCS. The HCS is calculated in the same way as the FCS; see 8.5.

#### 8.4.1.7 Information field

The information field may be any sequence of bytes. In the case of data frames (I and UI frames), it carries the MSDU.

#### 8.4.1.8 Frame check sequence (FCS) field

The length of the FCS field is two bytes. Unless otherwise noted, the frame checking sequence is calculated for the entire length of the frame, excluding the opening flag, the FCS and any start and stop elements (start/stop transmission). Guidelines to calculate the FCS are given in 8.5.

### 8.4.2 MAC addressing

#### 8.4.2.1 Use of extended addressing

As specified in ISO/IEC 13239:2002, 4.7.1, The address field range can be extended by reserving the first transmitted bit (low-order) of each address octet which would then be set to binary zero to indicate that the following octet is an extension of the address field. The format of the extended octet(s) shall be the same as that of the first octet. Thus, the address field may be recursively extended. The last octet of an address field is indicated by setting the low-order bit to binary one.

When extension is used, the presence of a binary "1" in the first transmitted bit of the first address octet indicates that only one address octet is being used. The use of address extension thus restricts the range of single octet addresses to 0x7F and for two octet addresses to 0...0x3FFF.

#### 8.4.2.2 Address field structure

The HDLC frame format type 3 (see 8.4.1.1) contains two address fields: a destination and a source HDLC address. Depending on the direction of the data transfer, both the client and the server addresses can be destination or source addresses.

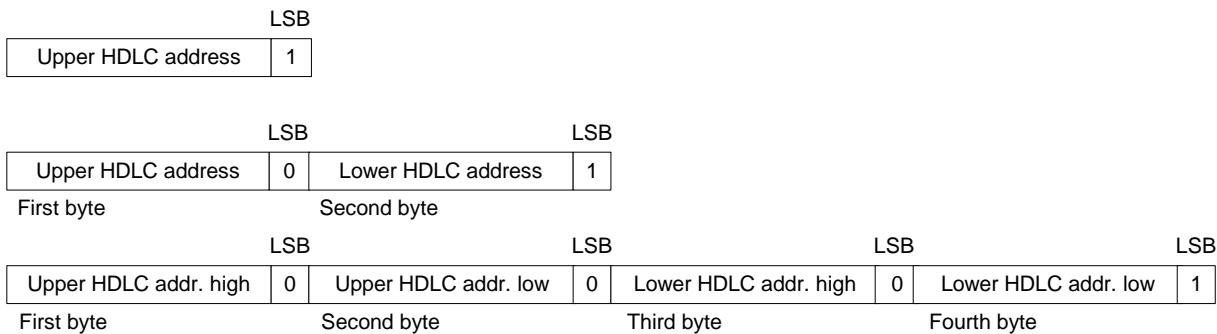
The client address shall always be expressed on one byte.

The server address – to enable addressing more than one logical device within a single physical device and to support the multi-drop configuration – may be divided into two parts:

- the upper HDLC address is used to address a Logical Device (a separately addressable entity within a physical device);
- the lower HDLC address is used to address a Physical Device (a physical device on the multi-drop).

The upper HDLC address shall always be present. The lower HDLC address may be omitted if it is not required.

The HDLC address extension mechanism applies to both parts. This mechanism specifies variable length address fields, but for the purpose of this protocol, the length of a complete server address field is restricted to be one, two or four bytes long, as shown on Figure 77. The server may support more than one addressing scheme. Individual, multicast and broadcast addressing facilities are provided for both the upper and the lower HDLC address.

**Figure 77 – Valid server address structures****8.4.2.3 Reserved special HDLC addresses**

The following special HDLC addresses are reserved:

**Table 13 – Table of reserved client addresses**

Reserved HDLC addresses	
No-station	0x00
Client Management Process	0x01
Public Client	0x10
	0x02 ... 0x0F
<i>Open for client SAP assignment</i>	0x11 ... 0xFF

**Table 14 – Table of reserved server addresses**

Reserved upper HDLC addresses		
	One byte address	Two byte address
No-station	0x00	0x0000
Management Logical Device	0x01	0x0001
Reserved for future use	0x02...0x0F	0x0002..0x000F
Open for server SAP assignment	0x10...0x7E	0x0010...0x3FFE
All-station (Broadcast)	0x7F	0x3FFF
Reserved lower HDLC addresses		
No-station	0x00	0x0000
Reserved for future use	0x01...0x0F	0x0001...0x000F
Open for server SAP assignment	0x10...0x7D	0x0010...0x3FFD
CALLING <sup>a</sup> ) Physical Device	0x7E	0x3FFE
All-station (Broadcast)	0x7F	0x3FFF

<sup>a</sup> The meaning of the CALLING Physical Device is discussed in 8.4.2.4.

In the table above, the effect of the address extension bits is not taken into account. Their use is illustrated with the following example:

Client HDLC Address = 0x3A = 00111010<sub>B</sub>

Server HDLC Address (using four bytes addressing)

lower HDLC Address = 0x3FFF = 0011111111111111<sub>B</sub> All-station (Broadcast) Address

upper HDLC Address = 0x1234 = 0001001000110100<sub>B</sub>

The address fields of the message shall contain the following octets:

Server address								Client address	
Upper HDLC high		Upper HDLC low		Lower HDLC high		Lower HDLC low		HDLC address	
LSB		LSB		LSB		LSB		LSB	
0	1	0	0	1	0	0	1	1	1
First byte		Second byte		Third byte		Fourth byte		Fifth byte	
Destination address								Source address	

**Figure 78 – Address example**

#### 8.4.2.4 Handling special addresses

The following MAC address types and specific MAC addresses are specified:

- individual addresses;
- group addresses;
- the All-station address;
- the No-station address;
- the CALLING Physical Device address;
- the Management Logical Device Address (the presence of this logical device is mandatory).

The following rules apply:

- group Address management is not within the Scope of this Technical Report;
- the Source Address field of a valid HDLC frame may not contain either the All-station or the No-station address. If an HDLC frame is received with it, it shall be considered as an invalid frame;
- only HDLC frames transmitted from the primary station towards the secondary station(s) may contain the All-station or the No-station in the Destination Address field;
- broadcast and multicast I frames shall be discarded;
- the P/F bit of messages with All-station, No-station or Group address in the Destination Address field shall be set to FALSE. UI frames containing an All-station, No-station or Group address with P == TRUE shall be discarded;
- the CALLING Physical Device address is a special address to support event reporting; see 8.4.5.4.7. It is reserved to reference the server station initiating a physical connection to the client station. It is not the station's own physical address; therefore no station shall be configured to have the CALLING Physical Address as its own physical address.

#### 8.4.2.5 Handling inopportune address lengths in the server

Frames received by the server may contain addresses with a different length than what is expected. In such cases, the following rules apply:

- as client addresses are specified to be one byte, frames that contain more than one byte in the source address field shall be discarded;
- destination addresses (DA) shall be handled according to Table 15.

**Table 15 – Handling inopportune address lengths**

Length of the DA field received	Length of the DA field expected	Behaviour
1 byte	2 bytes	The frame shall be discarded.
1 byte	4 bytes	The frame shall be discarded.
2 bytes	1 byte	The frame is not discarded only if the lower MAC Address is equal to the All-station address. In this case, it shall be given to the Logical Device(s) designated by the upper MAC Address field.
2 bytes	4 bytes	The value of the one-byte lower and upper MAC addresses received shall be converted into a two + two byte address. The frame shall be taken into account as if it was received using a 4-byte DA field.
4 bytes	1 byte	The frame is not discarded only if the both the lower and upper MAC Addresses are equal to the All-station address.
4 bytes	2 bytes	If the lower MAC Address is equal to the All-station address, the frame shall be accepted only if the upper MAC Address is also equal to the All-station address. If the lower MAC address is equal to the CALLING Physical Device address, the frame shall be accepted only if the upper MAC Address is equal to the Management Logical Device Address and the CALLING DEVICE layer parameter – see 8.4.5.4.8 – is set to TRUE. In any other case, the frame received shall be discarded.
3 or more than 4 bytes	N.A.	The frame shall be discarded.

#### 8.4.3 Command and response frames

##### 8.4.3.1 Selected repertoire and control field format

This Technical Report uses the UNC basic repertoire of commands and responses, extended with the UI commands and responses, as defined in ISO/IEC 13239. The encoding of the command/response control fields shall be modulo 8, as specified in 5.5 of ISO/IEC 13239), shown in Table 16.

**Table 16 – Control field bit assignments of command and response frames**

Command	Response	MSB	LSB
I	I	R R R P/F	S S S 0
RR	RR	R R R P/F	0 0 0 1
RNR	RNR	R R R P/F	0 1 0 1
SNRM		1 0 0 P	0 0 1 1
DISC		0 1 0 P	0 0 1 1
	UA	0 1 1 F	0 0 1 1
	DM	0 0 0 F	1 1 1 1
	FRMR	1 0 0 F	0 1 1 1
UI	UI	0 0 0 P/F	0 0 1 1

**RRR** is the receive sequence number N(R), **SSS** is the send sequence number N(S) and **P/F** is the poll/final bit.

**NOTE** In this notation, the bit order is inverted compared to the notation in ISO/IEC 13239. However, in both notations, the LSB is the first bit transmitted. For transmission order, see 8.4.4.2.2.

#### 8.4.3.2 Information transfer command and response

The function of the information command and response frame – the I frame – is to perform an information transfer. The I frame control field shall contain two sequence numbers:

- a) N(S), which shall indicate the sequence number associated with the I frame; and
- b) N(R), which shall indicate the sequence number (as of the time of transmission) of the next expected I frame to be received, and consequently shall indicate that the I frames numbered up to N(R) – 1 inclusive have been received correctly.

For data integrity reasons, in this profile, the default value of the maximum information field length – receive and maximum information field length – transmit HDLC parameters is 128 bytes. Other values may be negotiated at connection establishment time, see 8.4.5.3.1.

NOTE 1 In order to ensure a minimal performance, the master station should offer at least a max\_info\_field\_length\_receive of 128 bytes.

NOTE 2 The maximum value of the information field length is 2030 bytes.

The P/F bit that may be set to "1" or "0". Its use is explained in 8.4.4.1.

#### 8.4.3.3 Receive ready (RR) command and response

The Receive ready, RR, frame shall be used by a data station to:

- a) indicate that it is ready to receive an I frame(s); and
- b) acknowledge previously received I frames numbered up to N(R) - 1 inclusive.

When transmitted, the RR frame shall indicate the clearance of any busy condition that was initiated by the earlier transmission of an RNR frame by the same data station.

#### 8.4.3.4 Receive not ready (RNR) command and response

The Receive not ready, RNR, frame shall be used by a data station to indicate a busy condition, i.e. temporary inability to accept subsequent I frames. I frames numbered up to N(R) – 1 inclusive shall be considered as acknowledged. The I frame numbered N(R) and any subsequent I frames received, if any, shall not be considered as acknowledged; the acceptance status of these frames shall be indicated in subsequent exchanges.

#### 8.4.3.5 Set normal response mode (SNRM) command

The SNRM command shall be used to place the addressed secondary station in the normal response mode (NRM) where all control fields shall be one octet in length. The secondary station shall confirm acceptance of the SNRM command by transmission of a UA response at the first respond opportunity. Upon acceptance of this command, the secondary station send and receive state variables shall be set to zero.

When this command is actioned, the responsibility for all unacknowledged I frames assigned to data link control reverts to a higher layer. Whether the content of the information field of such unacknowledged I frames is reassigned to data link control for transmission or not is decided at a higher layer.

The SNRM command may contain an optional information field that is used for negotiation of data link parameters (see 8.4.5.3.1) and to carry user information transported transparently across the link layer to the user of the data link.

#### 8.4.3.6 Disconnect (DISC) command

The DISC command shall be used to terminate an operational or initialization mode previously set by a command. In both switched and non-switched networks, it shall be used to inform the addressed secondary station(s) that the primary station is suspending operation and that the secondary station(s) should assume a logically disconnected mode. Prior to actioning the command, the secondary station shall confirm the acceptance of the DISC command by the transmission of a UA response.

When this command is actioned, the responsibility for all unacknowledged I frames assigned to data link control reverts to a higher layer. Whether the content of the information field of such unacknowledged I frames is reassigned to data link control for transmission or not is decided at a higher layer.

An information field may be present in the DISC command.

#### **8.4.3.7 Unnumbered acknowledge (UA) response**

The UA response shall be used by the secondary station to acknowledge the receipt and acceptance of SNRM and DISC commands.

The UA response may contain an optional information field that is used for negotiation of data link parameters (see 8.4.5.3.2).

#### **8.4.3.8 Disconnected mode (DM) response**

The DM response shall be used to report a status where the secondary station is logically disconnected from the data link, and is, by system definition, in NDM.

The DM response shall be sent by the secondary station in NDM to request the primary/other combined station to issue a mode setting command, or if sent in response to the reception of a mode setting command, to inform the primary station that it is still in NDM and cannot action the mode setting command. An information field may be present in the DM response.

A secondary station in NDM shall monitor received commands to detect a respond opportunity in order to (re)transmit the DM response, i.e. no commands (other than UI commands) are accepted until the disconnected mode is terminated by the receipt of a mode setting command (SNRM).

#### **8.4.3.9 Frame reject (FRMR) response**

The FRMR response shall be used by the secondary station in an operational mode to report that one of the following conditions which is not correctable by retransmission of the identical frame resulted from the receipt of a frame without FCS error from the primary station:

- the receipt of a command or a response that is undefined or not implemented;
- the receipt of an I/UI command or response, with an information field which exceeded the maximum information field length which can be accommodated by the secondary/ combined station;
- the receipt of an invalid N(R) from the primary/combined station, i.e. an N(R) which identifies an I frame which has previously been transmitted and acknowledged or an I frame which has not been transmitted and is not the next sequential I frame awaiting transmission; or
- the receipt of a frame containing an information field when no information field is permitted by the associated control field.

The secondary station shall transmit the FRMR response at the first respond opportunity. An information field that provides the reason for the frame rejection shall be included (see 5.5.3.4.2 of ISO/IEC 13239).

#### **8.4.3.10 Unnumbered information (UI) command and response**

The UI command shall be used to send information to a secondary station(s) without affecting the V(S) or V(R) variables at any station. Reception of the UI command is not sequence number verified by the data link procedures. Therefore, the UI frame may be lost if a data link exception occurs during transmission of the command, or duplicated if an exception condition occurs during any reply to the command. There is no specified secondary station response to the UI command. The UI command may be sent independently of the mode of the data link station (NDM or NRM).

The UI response shall be used to send information (for example, status, application data, operation, interruption, or temporal data) to a primary/combined station without affecting the V(S) or V(R) variables at either station. Reception of the UI response is not sequence number verified by the data link procedures; therefore, the UI frame may be lost if a data link exception occurs during transmission of the UI response, or duplicated if an exception condition occurs during any reply to the UI response. The UI response may be sent during any mode of the data link station.

#### 8.4.4 Elements of the procedures

##### 8.4.4.1 Overview

When the physical link is established between the primary and the secondary stations, but there is no active data link channel established, both the client and the server side MAC sublayers are in the Normal Disconnected Mode, NDM; see 8.4.4.3. In this mode, no information or numbered supervisory frames are transmitted or shall be accepted. In this mode, the secondary station capability is limited to:

- accepting and responding to SNRM commands;
- accepting a UI command;
- transmitting a UI response at a respond opportunity;
- responding with a DM response to a received disconnect (DISC) command.

When an MAC connection is established, the MAC layer operates in the Normal Response Mode, NRM. The secondary station (the server) shall initiate data transmission only as a result of receiving explicit permission to do so from the primary station (the client). After receiving permission (POLL BIT == TRUE), the secondary station shall initiate a response transmission. The response transmission may consist of one or more frames, while maintaining active data link channel state (see 8.4.4.3.1). The last frame of the response transmission shall be explicitly indicated by the secondary station (FINAL BIT == TRUE). Following the indication of the last frame, the secondary station shall stop transmitting until explicit permission is received again from the primary station.

##### 8.4.4.2 Transmission considerations

###### 8.4.4.2.1 Transparency

ISO/IEC 13239:2002, 4.3 specifies multiple transparency mechanisms. For the purpose of this protocol, the non-basic frame format transparency mechanism has been selected, as it is specified in 4.3.4. When using the non-basic frame format with the frame format field, the length sub-field obviates the need for the bit or octet insertion methods to achieve transparency. Consequently, no control octet transparency (octet insertion) is used in this MAC sublayer operation.

###### 8.4.4.2.2 Order of bit and octet transmission

Addresses, commands, responses, sequence numbers and data link information within information fields shall be transmitted with the low-order bit first.

Fields, which carry values expressed in more than one octet, shall be transmitted with the most significant octet first. For example, if an address field is expressed on two octets with the value 0x1234, the most significant octet (0x12) will be transmitted first, followed by the least significant one (0x34). 16-bit HCS and FCS shall be transmitted to the line commencing with the coefficient of the highest term (corresponding to x15) also.

NOTE Subclause 8.5 gives an example for FCS calculation.

###### 8.4.4.2.3 Invalid frame

An invalid frame is defined as one that is not properly bounded by two flags, or one that is too short (that is shorter than seven octets between flags using the 16-bits FCS), or one in which octet framing is violated (for example an "0" bit occurs when the stop-bit is expected), or one containing either an All-station or No-station address in its Source Address field. Invalid frames shall be ignored.

#### 8.4.4.3 HDLC channel states

##### 8.4.4.3.1 Active HDLC channel state

A HDLC channel is in active state when the primary station or a secondary station is actively transmitting a byte of the frame or an inter-octet fill. In active state, the right to continue transmission shall be reserved.

#### 8.4.4.3.2 Abort sequence

The abort sequence is specified (see 5.1.1.2 of ISO/IEC 13239) as a two-octet sequence, starting with the control escape octet followed by a closing flag octet. Receipt of this sequence is interpreted as an abort and the receiving data station shall ignore the frame. In this Technical Report, the non-basic transparency is used; therefore the Abort Sequence HDLC feature cannot be used.

#### 8.4.4.3.3 Start/stop transmission inter-octet time-out

The inter-octet time-out ( $T_{io}$ ) for start/stop transmission is an optional time-out used for recovering from situations in which excessive periods of time elapse between transmitted octets within a frame. This time-out function (or equivalent) only applies to a frame that is being received. The time-out function (or equivalent) is started once the stop bit of an octet is detected and stopped upon receipt of the start bit of the next octet or when the time-out function (or equivalent) runs out. Whenever this time-out occurs in the receiver, the end of the actually received frame shall be assumed and the data stream shall be scanned for the next opening flag sequence.

NOTE The value of  $T_{io}$  depends on the media used. For PSTN connection  $T_{io} = 25$  ms.

#### 8.4.4.3.4 Idle HDLC channel state

A Data Link channel is in idle state when a continuous mark-hold condition persists for a system specific period of time ( $T_{idle}$ ).

NOTE The value of  $T_{idle}$  depends on the media used. For PSTN connection  $T_{idle} = 25$  ms.

### 8.4.5 HDLC channel operation – Description of the procedures

#### 8.4.5.1 General

For the purpose of this Technical Report, unbalanced connection-mode data link operation has been selected. An unbalanced data link involves one primary station and one or more secondary station(s). The primary station shall be ultimately responsible for overall data link error recovery.

ISO/IEC 13239:2002, 5.2 defines three operational and three non-operational modes. For the purpose of this Technical Report, the NRM and the NDM modes have been selected.

Each data station shall check for the correct receipt of the I frames it has sent to the remote data station by checking the N(R) of each numbered information and supervisory frames received.

#### 8.4.5.2 Data station characteristics

The primary station is responsible for:

- setting up the data link and disconnecting the data link;
- sending information transfer, supervisory and unnumbered commands; and
- checking received responses.

The secondary station shall be responsible for:

- checking received commands;
- sending information transfer, supervisory and unnumbered responses as required by the received commands.

#### 8.4.5.3 Procedures for setting up and disconnecting the data link

##### 8.4.5.3.1 Setting up the data link

The primary station shall initialize the HDLC link with the secondary station by sending an SNRM command and shall start a response time-out function (see 8.4.5.6.1). The addressed secondary station, upon receiving the SNRM command correctly, shall send the UA response at its first opportunity, and shall set its send and receive state variables to zero. If the UA response is received correctly, the HDLC link set up to the addressed secondary station is complete, and the primary station shall set its send and receive state variables relative to that secondary station to zero and shall stop the response time-out function. If, upon receipt of an SNRM command the secondary station determines that it cannot enter the indicated mode, it shall send the DM response. If the DM response is received correctly, the primary station shall stop the response time-out function.

If the SNRM command, UA response or DM response is not received correctly, it shall be ignored. The result will be that the primary station's response time-out function will run out, and the primary station may re-send the SNRM command and restart the response time-out function.

This action may continue until an UA or DM response has been received correctly or until the SNRM frame is transmitted MAX\_NB\_OF\_RETRIES times (see 8.4.5.6.2). Any further recovery action takes place at a higher layer.

#### 8.4.5.3.2 HDLC parameter negotiation during the connection phase

The SNRM/UA message exchange allows not only the establishment of the connection, but also to negotiate some data link parameters. For the purpose of this protocol, two negotiable parameters specified in ISO/IEC 13239 have been selected:

- the Maximum information field length parameter; and
- the Window size parameter.

The default value of the maximum information field length parameter is 128 bytes. The maximum value depends on the quality of the physical channel.

The default value of the Window size is 1. The maximum value is 7.

Besides these negotiable HDLC parameters, the Information Field of an SNRM frame may also contain a User data subfield.

Including other parameters in the Information Field implies the rejection of the SNRM frame. In this case, a DM frame (with "Incorrect Parameter list within the Information Field of the SNRM frame") shall be sent to the primary station.

The negotiation of these parameters takes place using the optional Information Field of the SNRM / UA frames.

When it is present, the first two bytes – Format Identifier (0x81) and Group Identifier (0x80) – shall also always be present. On the other hand, any (one or more) of the negotiable parameters may be absent. The absence of a particular parameter (PV=0 or PI/PL/PV missing, see 5.5.3.1.2 of ISO/IEC 13239) shall be interpreted to mean default values.

The absence of the Information Field shall be interpreted to mean default values for each parameter.

Example of an information field encoding (the parameter values representing the default):

81	80	12	05	01	80	06	01	80	07	04	00	00	00	01	08	04	00	00	00	01
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

where:

- 0x81 format identifier;
- 0x80 group identifier;
- 0x12 group length (18 octets);
- 0x05 parameter identifier (maximum information field length – transmit);
- 0x01 parameter length (1 octet);
- 0x80 parameter value;
- 0x06 parameter identifier (maximum information field length – receive);
- 0x01 parameter length (1 octet);
- 0x80 parameter value;
- 0x07 parameter identifier (window size, transmit);
- 0x04 parameter length (4 octets);
- 0x00 parameter value (high byte of value);
- 0x00 parameter value;
- 0x00 parameter value;
- 0x01 parameter value (low byte of value);
- 0x08 parameter identifier (window size, receive);
- 0x04 parameter length (4 octets);
- 0x00 parameter value (high byte of value);
- 0x00 parameter value;
- 0x00 parameter value;
- 0x01 parameter value (low byte of value).

In the case, when for the *maximum information field length – transmit* and *maximum information field length – receive* parameters values above 128 (0x0080) are allowed – this is the case when version 1 of the “IEC HDLC setup” class is used, see DLMS UA 1000-1 – the group length will be 0x14 (20 octets) and the parameter length of parameters 5 and 6 will be 0x02. An example is shown below:

81	80	14	05	02	00	80	06	02	00	80	07	04	00	00	00	01	08	04	00	00	01
----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----	----

If the secondary station receives a correct SNRM frame, and the requested connection can be accepted, it shall respond with a UA frame. This UA frame shall carry the result of the HDLC parameter negotiation. The result shall be calculated by the secondary station by choosing the smaller value between the proposed value of a parameter (sent with the SNRM frame) and the value supported by the secondary station. An example is given in Table 17.

**Table 17 – Example for parameter negotiation values with the SNRM/UA frames**

Parameter / value proposed (by the SNRM frame)	Parameter / value supported by the secondary station	Result (transmit/receive direction from the point of view of the secondary station)
Max_info_field_length-transmit (0x05) / 0x80 (128 <sub>D</sub> )	Max_info_field_length-receive (0x06) / 0x40 (64 <sub>D</sub> )	Max_info_field_length-receive (0x06) / 0x40 (64 <sub>D</sub> )
Max_info_field_length-receive (0x06) / 0x80 (128 <sub>D</sub> )	Max_info_field_length-transmit (0x05) / 0x80 (128 <sub>D</sub> )	Max_info_field_length-transmit (0x05) / 0x80 (128 <sub>D</sub> )
Window size-transmit (0x07) / 0x0001 (1 <sub>D</sub> )	Window size-receive (0x08) / 0x0007 (7 <sub>D</sub> )	Window size-receive (0x08) / 0x0001 (1 <sub>D</sub> )
Window size-receive (0x08) / 0x0007 (7 <sub>D</sub> )	Window size-transmit (0x07) / 0x0007 (7 <sub>D</sub> )	Window size-transmit (0x07) / 0x0007 (7 <sub>D</sub> )
NOTE For reasons of communication efficiency, the value of the parameter Max_info_field_length-receive proposed by the primary station shall be at least as big as the value of the parameter of the Max_info_field_length-transmit supported by the secondary station.		

The SNRM frame in the case above shall include an Information Field, but only the presence of the ‘Window size – receive’ parameter is mandatory, because the values of the other parameters are default values.

The UA frame shall also include an Information Field. It shall contain the ‘Max\_info\_field\_length-receive’ (0x40), and the ‘Window size – transmit’ (0x07) parameters. The other two parameters are not necessarily present, because the other parameters are default values.

After the successful exchange of SNRM/UA frames with the above parameters, both sides shall consider that the HDLC parameters for the connection established are the values shown in the “Result” column of Table 17.

#### **8.4.5.3.3 Disconnecting the data link**

The primary station shall disconnect the MAC link with secondary station by sending a DISC command and shall start a response time-out function. The addressed secondary station, upon receiving the DISC command correctly, shall send a UA response at its first opportunity and shall enter the NDM. If, upon receipt of the DISC command, the addressed secondary station is already in the disconnected mode, it shall send a DM response. The primary station, upon receiving a UA or DM response to a DISC command, shall stop the response time-out function.

In the case of a multi-point configuration, the UA response from the secondary stations shall not interfere with one another. The mechanism to avoid overlapping responses to the DISC command using a group address or the all-station address is not within the Scope of this Technical Report. See also 8.4.5.4.6.

If the DISC command, UA response or DM response is not received correctly, it shall be ignored by the receiving station. This will result in the expiry of the primary station’s response time-out function, and the primary station may re-send the DISC command and restart the response time-out function.

This action may continue until a UA or DM response has been received correctly or until the DISC frame is transmitted MAX\_NB\_OF\_RETRIES times. Any further recovery action takes place at a higher layer.

#### **8.4.5.4 Procedures for data exchange**

##### **8.4.5.4.1 General**

The receipt of a MA-DATA.request primitive will cause the MAC sublayer to transmit the received Data to the peer MAC sublayer using HDLC protocol procedures. Transferring Information frames is only allowed when MAC Connection is already established and the MAC sublayer is in operational mode (NRM). In the case of Frame\_type == I-COMPLETE (Information) this Data will be transmitted in one or several information frames (I frames). Sequence numbering and acknowledgement procedures of HDLC shall be applied. If necessary, the Data will be divided into sub-frames (segmentation) prior to transmission. The peer MAC layer has to recombine these sub-frames after reception. A special procedure is specified to allow transmitting long messages from the server to the client. This procedure is using the frame types I-FIRST-FRAGMENT, I-FRAGMENT, I-LAST-FRAGMENT, and it is described in 8.4.5.4.5.

In the case of Frame\_type == UI the received Data will be transmitted in a UI frame. In this case, the size of the Data shall fit in one HDLC frame. UI frames basically serve for broadcasting/multicasting, but may also be used for event reporting. UI frames may be sent both in operational mode (NRM) and in non-operational mode (NDM).

##### **8.4.5.4.2 Exchange of information frames**

Information frame exchange is specified in detail in 6.11.4.2 of ISO/IEC 13239.

In summary, as the NRM is used, the secondary station shall initiate transmission only as the result of receiving explicit permission to do so from the primary station; the primary station initiates each data exchange. The primary station shall set the poll bit to “1” in the last frame of a transmission to solicit response frames from the secondary station. The secondary station shall set the final bit to indicate the last frame of its response transmission (see 5.4.3 in ISO/IEC 13239).

I frames containing N(S) send and N(R) receive sequence numbers are used for confirmed information transfer. I frames shall be acknowledged by the receiver. Several I frames may be linked together; the poll/final bit indicates the last frame of such a sequence.

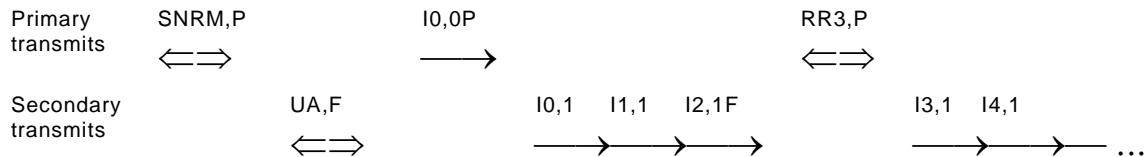
A time-out function shall be used in the primary station to monitor the responses of the secondary station.

The following examples show typical exchanges of frames in NRM with TWA transmission (see 6.11.4.5 of ISO/IEC 13239). More examples can be found in ISO/IEC 13239, Annex B.

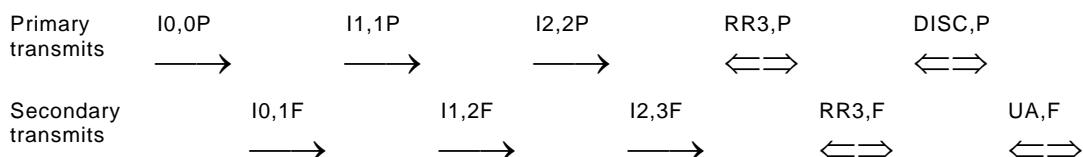
The following notation is used:

→ frame with info, ↔ frame without info, P / F poll/final bit, I / RR: Frame type, "n, n" values of N(S) and N(R)

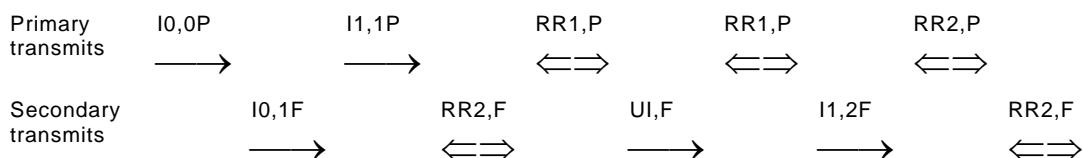
**Example 1: Start-up procedure and information transfer without transmission errors (window size=3):**



**Example 2: Information transfer (window size 1) and closing procedure:**



**Example 3: Information transfer with intermediate UI frame:**



#### 8.4.5.4.3 Window size considerations

The HDLC standard allows transferring more than one frame in a sequence before an acknowledgement is due. The send and receive sequence numbers allow acknowledgement with the information of how many frames have been correctly received. The maximum number of consecutive frames is referred to as the *window size*.

If window size > 1 is used, consecutive frames may be linked together. In this case, a single flag is used as both the closing flag for one frame and the opening flag for the next frame (see Figure 75).

If window size > 1 is used and frames are not linked together, a time-out between consecutive frames has to be observed (see 8.4.4.3.3).

#### 8.4.5.4.4 Segmentation

In the frame format field, the segmentation subfield of 1 bit follows the format type subfield and if present, reduces the length subfield by one bit. The field is used as follows (see 4.9.3 of ISO/IEC 13239):

- all MSDUs transmitted shall have the segmentation algorithm applied. The algorithm shall be applied to MSDUs which fit in a single HDLC frame or those which must be transmitted as a sequence of HDLC frames;
- the final HDLC frame of an MSDU shall be sent with segmentation subfield = 0;
- when MSDUs must be transmitted in multiple HDLC frames, all except the final HDLC frame of the MSDU shall be sent with their segmentation sub-field = 1;
- the HDLC window sequence numbers guarantee that all segments are sent/received in order and that lost segments can be detected.

#### 8.4.5.4.5 Transferring long MSDUs from the server to the client

As servers are often embedded systems with relatively poor memory resources, a special mechanism is specified to transmit 'long' MSDUs from the server to the client.

**NOTE** From the implementation point of view, an MSDU is considered to be 'long' when the server does not have enough room to allocate a buffer, which may contain the complete MSDU.

In order to transmit transparently (from the point of view of the peer client MAC layer) these long Service User layer PDUs, the server service user layer shall be able to segment the complete PDU and send the first segment of it by invoking the DL-DATA.request primitive with Frame\_type == I-FIRST-FRAGMENT.

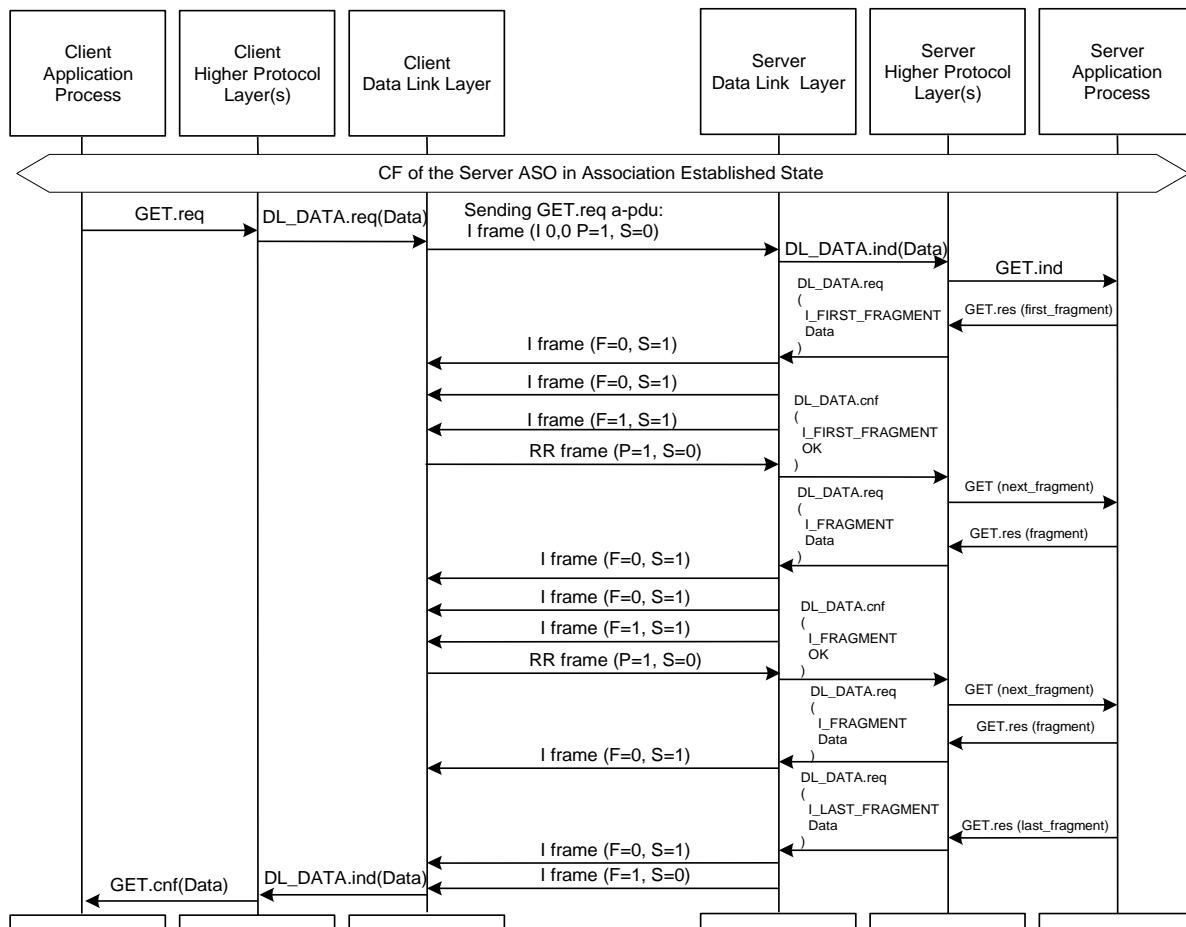
The server MAC sublayer shall send the received MSDU within as many HDLC frames as necessary, but it shall send the last frame with the Segmentation bit set to 1. When this last frame is acknowledged by the client MAC sublayer (with a RR frame), the server MAC sublayer shall generate an MA-DATA.confirm primitive with Frame\_type = I-FIRST-FRAGMENT.

When the server Service User layer receives the DL-DATA.confirm primitive, it shall send the next segment of the long PDU by invoking the DL-DATA.request primitive with Frame\_type == I-FRAGMENT. HDLC frames containing this segment shall be transmitted similarly to the frames of the first segment: even the last HDLC frame shall be sent with Segmentation bit = 1, and on the receipt of the acknowledgement of this last frame a DL-DATA.confirm primitive shall be generated with Frame\_type == I-FRAGMENT.

The following segments of the long Service User layer PDU shall be transmitted in the same way – until the last one, which shall be transmitted using a DL-DATA.request primitive with Frame\_type == I-LAST-FRAGMENT. The final HDLC frame of that last fragment shall be transmitted with the Segmentation bit = 0, meaning that the complete PDU has been sent – and received at the client side.

The client side shall generate an MA-DATA.indication primitive only when this final HDLC frame (with the Segmentation bit = 0) is received: the fragmentation procedure provided by the server is not at all visible at the client side. That is the reason that this procedure is considered to be transparent.

Figure 79 shows the corresponding message sequence chart for the GET.request AL service. The Read.request service is treated in the same way.



**Figure 79 – MSC for long MSDU transfer in a transparent manner**

#### 8.4.5.4.6 Multi- and broadcasting

Multicasting and broadcasting are possible using UI frames. In the present environment, this is allowed only for the clients – servers are not allowed to send frames with broadcast or multicast address in the Destination\_Address field.

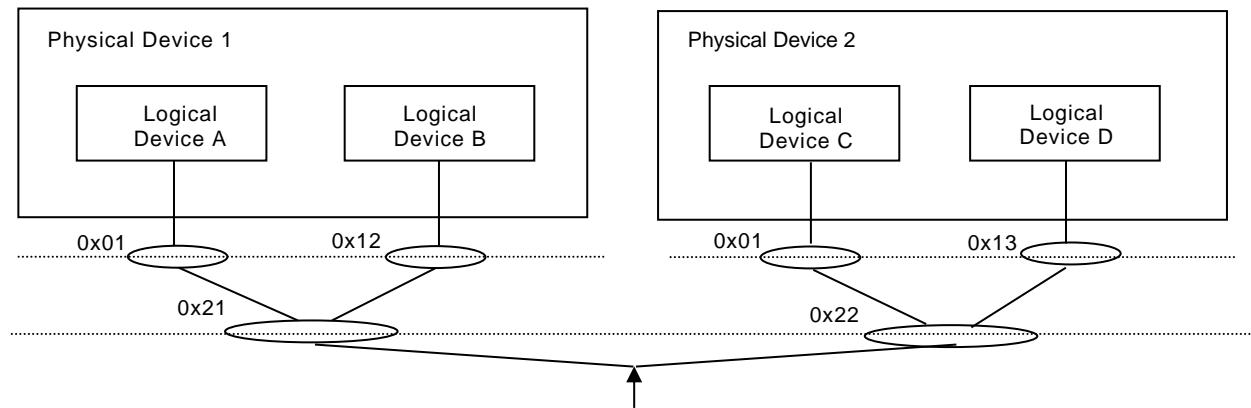
UI frames shall be reported to the service user layer if the Destination Address of the frame designates one of the local MSAPs, if it is an existing local group address, or if it is the broadcast MAC address.

Only UI (and DISC) messages may serve as broadcast or multicast messages; all other message types with any broadcast or multicast address in the Destination\_Address field shall be discarded by the server MAC sublayer. Broadcast and multicast UI messages shall always be sent with Poll bit == FALSE. Broadcast and multicast UI frames with Poll == TRUE shall be discarded.

**NOTE** When the data link layer specified in this Technical Report is used together with the DLMS/COSEM AL specified in Clause 9, releasing of an AA is done by disconnecting the supporting data link layer connection. To release non-pre-established multiple AAs, the DISC command may also be sent with a broadcast or multicast address. The mechanism used to avoid overlapping responses to the disconnect (DISC) command using a group address or the All-station address is not within the scope of this Technical Report.

Broadcast and multicast are allowed both to the logical devices within a physical device, using the upper HDLC address and to physical devices, using the lower HDLC address.

Figure 80 shows an example with two physical devices, each of them including two logical devices.

**Figure 80 – Example configuration to illustrate broadcasting**

As shown on Figure 80, the MAC addresses of these devices are the following:

**Table 18 – Summary of MAC addresses for the example**

	<b>Upper MAC address (logical device)</b>	<b>Lower MAC address (physical device)</b>
Logical Device A	0x01	0x21
Logical Device B	0x12	0x21
Logical Device C	0x01	0x22
Logical Device D	0x13	0x22

Messages with a broadcast address at any MAC address level shall be handled as specified in Table 19.

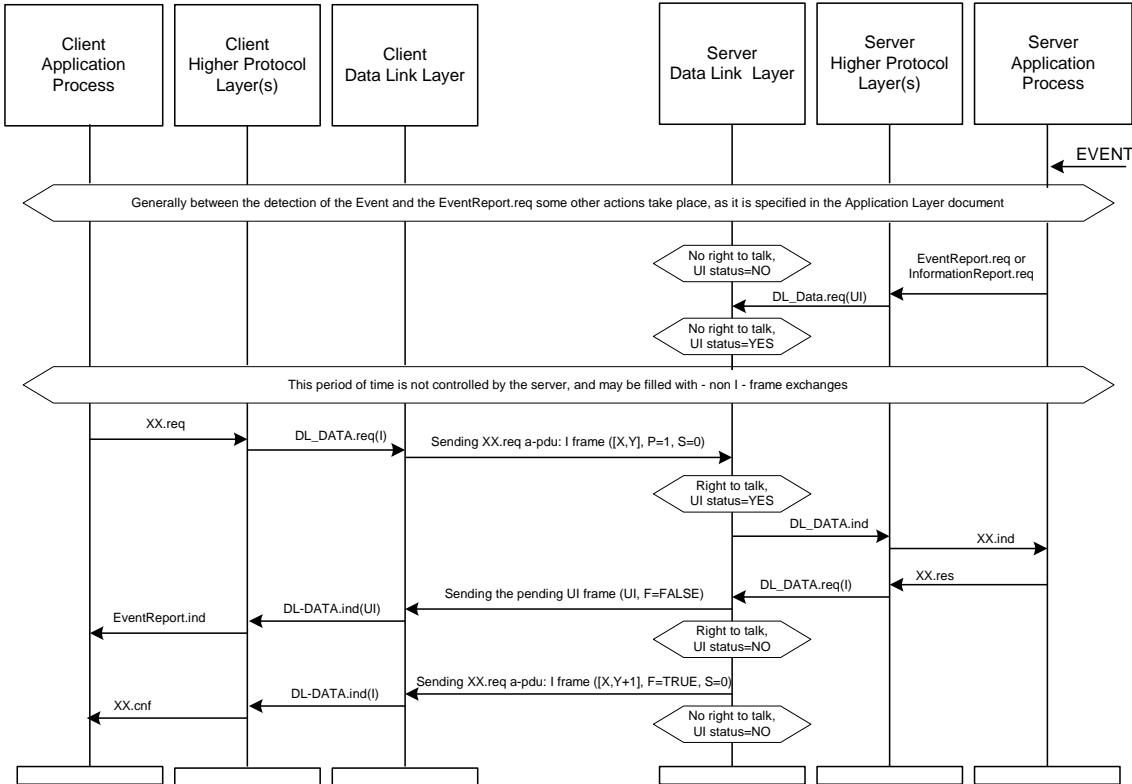
**Table 19 – Broadcast UI frame handling**

<b>Lower MAC address (Physical device address)</b>	<b>Upper MAC address (Logical device address)</b>	<b>UI message handling</b>
Individual (e.g. 0x21)	Individual (e.g. 0x01)	The incoming UI frame shall be sent to the individually addressed logical device within the individually addressed physical device. In the example, it is logical device A. The MAC sublayer of physical device 2 shall discard the received frame.
Individual (e.g. 0x22)	Broadcast 0x(7F)	The incoming UI frame shall be sent to all logical devices within the individually addressed physical device. In the example, the message shall be sent to logical devices C and D. The MAC sublayer of physical device 1 shall discard the received frame.
Broadcast (0x7F)	Individual (e.g. 0x01)	The incoming UI frame shall be sent to all individually addressed logical devices in all physical devices. In the example, the message shall be sent to logical devices A and C
Broadcast (0x7F)	Broadcast (0x7F)	The incoming UI frame shall be sent to all logical devices in all physical devices. In the example, the message shall be sent to logical devices A, B, C, and D

#### 8.4.5.4.7 Sending an UI frame from the server to the client

This clause discusses a situation where a server MAC layer receives a MA-DATA.request service invocation with Frame\_type = UI.

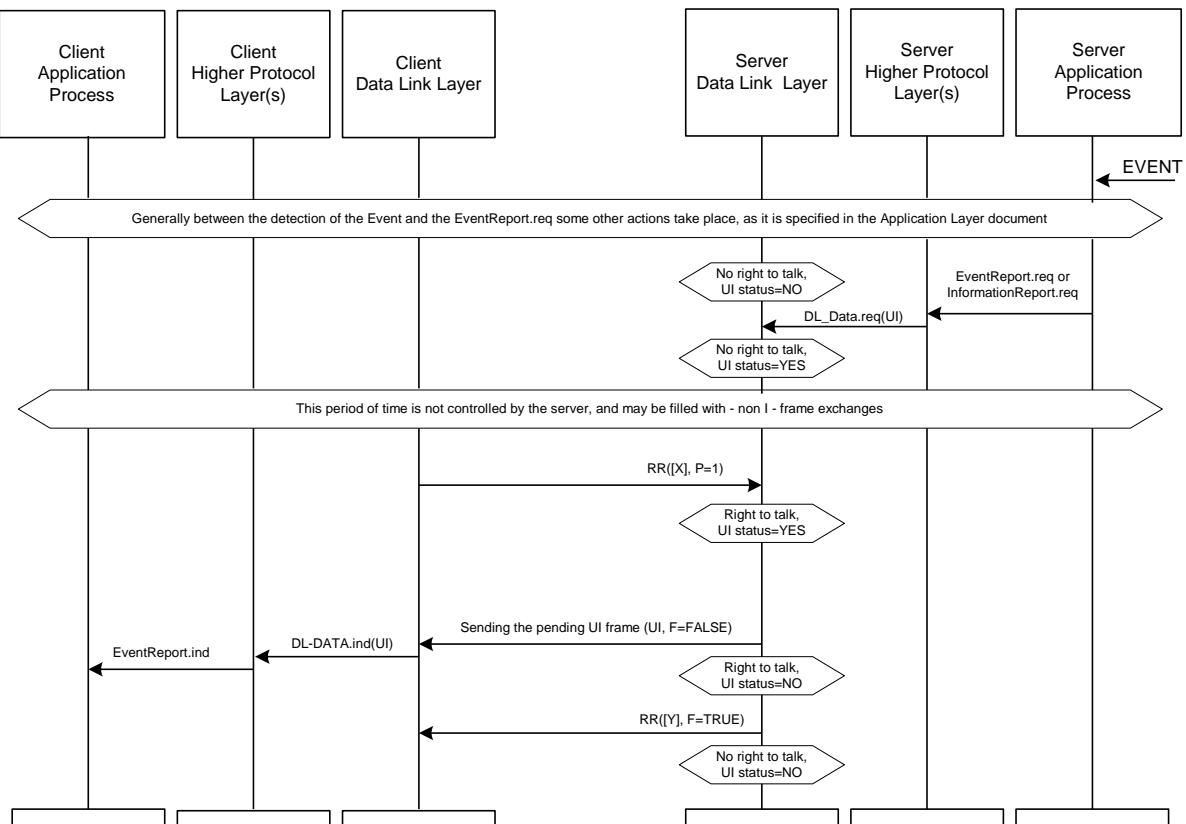
By its nature, this request may happen at any moment, in an asynchronous manner. (It is not the result of a previous MA-DATA.indication, thus it is non-solicited.) Therefore, when it happens, the server MAC layer generally has no right to send it out immediately (“no right to talk”). Consequently, this UI packet shall be stored in the MAC layer, waiting for the opportunity to be sent out.



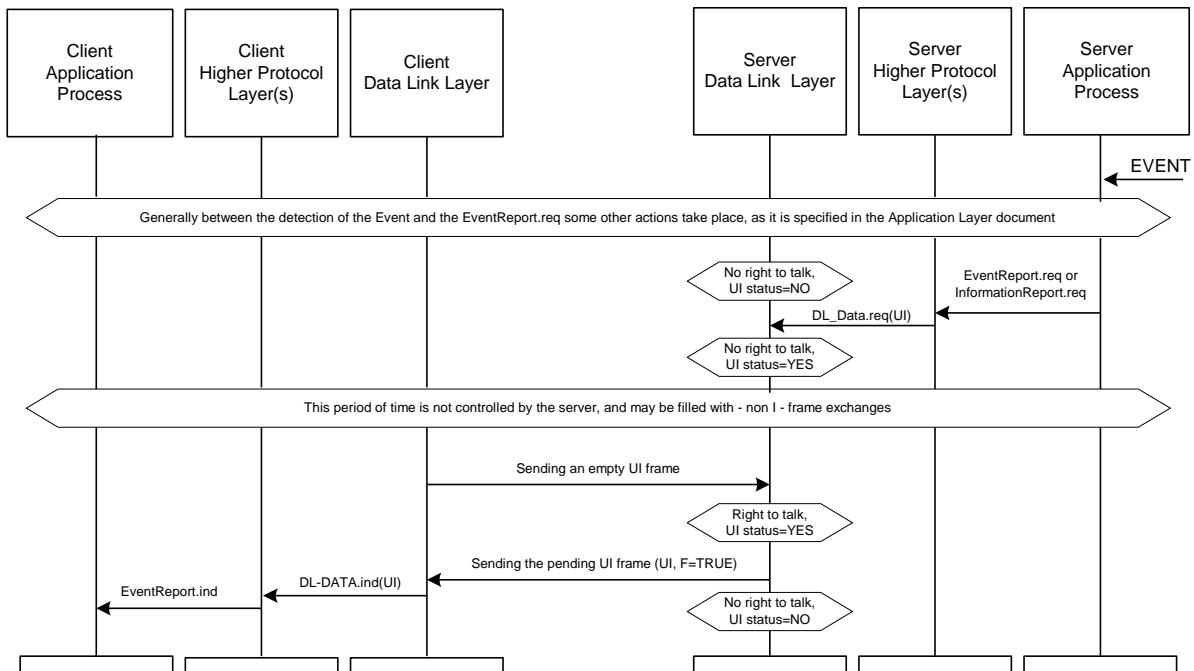
**Figure 81 – Sending out a pending UI frame with a .response data**

Pending UI frames can be sent on the following conditions:

- When an MA-DATA.request service primitive with frame\_type I-COMPLETE, I-FIRST-FRAGMENT, I-FRAGMENT, or I-LAST-FRAGMENT is received. The pending UI frame(s) shall be sent out just before the last I frame, which shall contain the closing Final=TRUE bit of the transmission, see Figure 81;
- In receipt of a RR frame with P=1. If the server MAC layer has no pending I or UI frame when a RR frame is received, normally it shall respond with another RR frame, just to give back the “right to talk” to the HDLC primary station. When there is a pending UI frame, this UI frame shall be sent out before the normal response frame, see Figure 82. For the use of the R frame, see 8.4.3.3;
- In receipt of a UI frame with P=1 and with zero length information field (empty UI frame). The receipt of this frame shall make the server data link layer send out all pending UI frames. The last UI frame shall be sent out with F=TRUE, see Figure 83.



**Figure 82 – Sending out a pending UI frame with a response to a RR frame**



**Figure 83 – Sending out a pending UI frame on receipt of an empty UI frame**

#### 8.4.5.4.8 Handling the CALLING device physical address

It is possible that the server initiates a physical connection establishment to the client device. Once this is established, the server can place a UI message in the MAC layer, which sends it out on one of the conditions discussed in 8.4.5.4.7.

As the client does not know whether the incoming call has come from a single device or from a multi-drop configuration, it shall use CALLING Physical Device address in place of the MAC Physical Destination Address parameter. If the client wishes to send an SNRM frame to the server, the frame shall contain the following parameters:

Source Address:	Client Management Process HDLC Address (0x01)
Destination Address	Lower MAC Address:      CALLING Physical Device Address (0x7E) Upper MAC Address:      Management Logical Device Address (0x01)
Poll bit:	TRUE
Information Field:	Negotiable HDLC Layer Parameters (if any)

Only the initiator server device shall respond to this SNRM frame.

When the initiator is a single device, it does not necessarily have its own proper physical device address. Nevertheless, it shall accept the incoming SNRM frame including the CALLING Physical Device address (a special lower MAC Address, reserved for this purpose, see Table 14).

A problem might occur when the initiator of the call was a server of a multi-drop configuration: all servers of the multi-drop will receive the incoming SNRM frame, and all servers shall recognize that it is addressed to the CALLING Physical Device. In order to avoid potential conflicts, and to correctly handle the CALLING Physical Device address, the lower MAC layer shall contain a CALLING DEVICE (Boolean) layer parameter, with default value = FALSE. Once the device's AP initiates a call, it shall set this layer parameter to TRUE.

NOTE Single meters may also use the CALLING DEVICE layer parameter. In this case, it shall always be set to TRUE.

When the MAC layer receives a HDLC frame with the CALLING device address in the 'Physical MAC Address' portion of the Destination Address field, it shall check the value of the CALLING DEVICE layer variable, and if it is FALSE, it shall discard the incoming frame. On the other hand if CALLING DEVICE = TRUE, the MAC layer shall consider that it was addressed to that device, and shall generate the DL-CONNECT.indication primitive to the upper layer. Using this layer variable ensures that only the originator server shall respond to the received SNRM frame.

When the MAC sublayer of the initiator device recognizes that the incoming SNRM frame has been addressed to that device via the CALLING Physical Device address, it shall handle this SNRM exactly as if it was addressed to this device via its proper physical address: depending on the acceptability of the requested MAC connection it shall respond either with a UA or with a DM frame. The Lower MAC Address sub-field of the source address field of this UA or DM frame shall contain the proper Physical Address of the responding device, not the CALLING Physical Device address.

When the current session terminates – a PH-ABORT.indication primitive is received – the MAC layer shall set the value of the CALLING DEVICE variable to FALSE.

#### 8.4.5.5 Exception recovery

##### 8.4.5.5.1 Response time-out

In the primary station, a time-out function is used to monitor the responses of the secondary station (see 6.11.4.3 of ISO/IEC 13239).

The response timer is started after transmission of a frame with the poll bit set to "1". It is restarted after receipt of an error-free frame with the final bit set to "0". It is stopped after receipt of an error-free frame with the final bit set to "1".

If a time-out occurs, the primary station shall retransmit the last frame it has sent. If the last frame sent was an information frame, this I frame shall not be retransmitted but a RR frame shall be sent in order to poll the secondary station and resynchronize I frame numbering.

The duration of the time-out function and the maximum number of retries are defined in 8.4.5.6.

#### 8.4.5.5.2 FCS and HCS error

Frames that have no HCS (only an FCS, frames without information field) shall be treated as described in 4.2.6 of ISO/IEC 13239. Frames received with an HCS error are treated as described in 5.6.3 of ISO/IEC 13239. Furthermore, all consecutive frames that are directly concatenated have to be discarded (no frame resynchronization possible because the length field of the erroneous frame has to be ignored). Frames received with an FCS error, but with an error-free header (no HCS error), shall not be accepted by the receiver, except for examination of the length field, the state of the P/F bit and the value of the N(R) field.

#### 8.4.5.5.3 N(S) sequence error

Sequence errors shall be handled as defined in 5.6.2 of ISO/IEC 13239. Poll/final bit recovery shall be used (5.6.2.1 of ISO/IEC 13239).

#### 8.4.5.5.4 Command/response frame rejection

A command/response rejection exception condition shall be handled as specified in 5.6.4 of ISO/IEC 13239. The secondary station shall send a FRMR response with appropriate diagnostics information. On receipt of a FRMR response, the primary station shall at least issue a SNRM command before continuing with information exchange.

#### 8.4.5.5.5 Busy

Busy conditions shall be treated as specified in 5.6.1 of ISO/IEC 13239.

### 8.4.5.6 Time-outs and other MAC sublayer parameters

#### 8.4.5.6.1 Time-out 1: Response time-out (TO\_WAIT\_RESP)

This time-out function is provided only by the MAC sublayer of the primary station (client) – and is the same for all command (SNRM, DISC) and numbered information (I) frames. The maximum time waited by the primary station for the return frame from the secondary station before retransmission must be chosen for example as:

$$\text{TO_WAIT_RESP} > \text{RespTime} + 2 * \text{MaxTxTime}$$

where RespTime represents the theoretical response time of the secondary station and MaxTxTime the maximum time for transmission of a frame.

#### 8.4.5.6.2 Layer Parameter 1: Maximum number of retries (MAX\_NB\_OF\_RETRIES)

If no response frame is received during TO\_WAIT\_RESP time-out, the client MAC sublayer will repeat the transmission of the command frame MAX\_NB\_OF\_RETRIES times. This parameter is provided only by the client MAC sublayer. If no response is received after the last time-out, the MAC sublayer will generate a .confirm service indicating the reason (NOK\_MAX\_NB\_RETRIES). In this case, the user layer shall request shutting down of the MAC Connection by invoking the MA-DISCONNECT.request primitive.

NOTE 1 This parameter does not apply on the unnumbered information (UI) frames.

NOTE 2 Shutting down the MAC connection is not done by the client MAC sublayer itself.

#### 8.4.5.6.3 Time-out 2: Inactivity time-out

This time-out is re-started each time when an octet is sent or received to/from the PHL. If the Inactivity time-out runs out, the data link layer shall generate a DL-LM\_EVENT.indication primitive – see 8.6.2.7 – signalling that no character has been sent / received during that period, and re-start the inactivity time-out. The data link layer shall be disconnected.

#### 8.4.5.6.4 Time-out 3: Inter-frame time-out

The maximum permitted time between the stop bit of a character (octet) and the start bit of the next character within a frame ( $T_{in}$ ) shall be selected to meet the requirements of the physical medium used. Whenever this time-out occurs in the receiver, the end of the actually received frame shall be assumed.

NOTE The inter-octet time-out, defined in 8.4.4.3.3, is the same as the inter-frame time-out.

186/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

#### 8.4.5.6.5 Maximum information field length

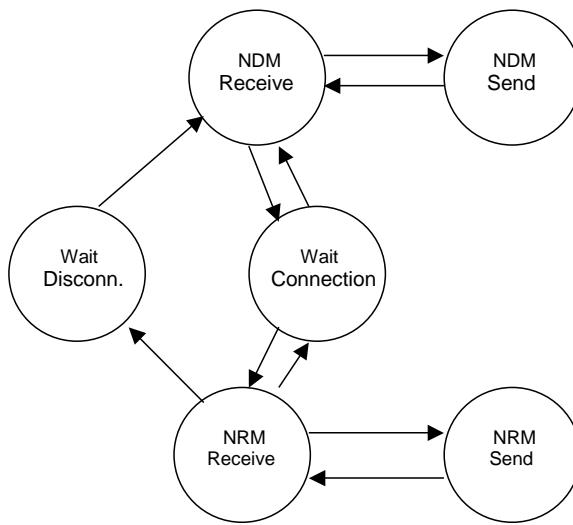
The default value of the Maximum information field length parameter is 128. This parameter can be negotiated upon data link layer connection establishment.

#### 8.4.5.6.6 Window size

The default value of the Window size parameter is 1. This parameter can be negotiated upon data link layer connection establishment.

#### 8.4.5.7 State transition diagram for the server MAC sublayer

Figure 84 shows a simplified state transition diagram for the server MAC sublayer.



**Figure 84 – State transition diagram for the server MAC sublayer**

### 8.5 FCS calculation

#### 8.5.1 Test sequence for the FCS calculation

NOTE The test sequence presented here can be found in the 1988 CCITT Blue Book X.1 – X.32, Appendix I.

The example presented here shows the proper FCS value for a two-byte frame consisting of 0x03 and 0x3F. The complete resulting frame is 0x7E 0x03 0x3F 0x5B 0xEC 0x7E.

$\vee$ – first bit transmitted 0111 1110 1100 0000 flag	1111 1100 address	1101 1010 0011 0111 control	0111 1110 FCS	last bit transmitted – $\vee$ flag
---	----------------------	--------------------------------	------------------	---------------------------------------

In the test sequence, the following rules (according to ISO/IEC 13239) are considered:

- the FCS is calculated considering the bit order as transmitted on the channel;
- for the address field, the control field and all the other fields (including the data, except the FCS) the low order bit (of each byte) is transmitted first (this rule is automatically followed by the UART);
- for the FCS the coefficient of highest term (corresponding to x15) is transmitted first.

#### 8.5.2 Fast frame check sequence (FCS) implementation

The following example implementation of the 16-bit FCS calculation is derived from RFC 1662.

The FCS was originally designed with hardware implementations in mind. A serial bit stream is transmitted on the wire, the FCS is calculated over the serial data as it goes out and the complement of the resulting FCS is appended to the serial stream, followed by the Flag Sequence.

The receiver has no way of determining that it has finished calculating the received FCS until it detects the Flag Sequence. Therefore, the FCS was designed so that a particular pattern results when the FCS operation passes over the complemented FCS. A good frame is indicated by this "good FCS" value.

### 8.5.3 16-bit FCS computation method

The following code provides a table lookup computation for calculating the FCSequence as data arrives at the interface.

```

/*
 * u16 represents an unsigned 16-bit number. Adjust the typedef for
 * your hardware.
 *
 * Drew D. Perkins at Carnegie Mellon University.
 *
 * Code liberally borrowed from Mohsen Banan and D. Hugh Redelmeier.
 *
 */
typedef unsigned short u16;
/*
 *
 * FCS lookup table as calculated by the table generator.
 *
 */

static u16 fcstab[256] = {
    0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
    0x8c48, 0x9dc1, 0xaf5a, 0xbcd3, 0xca6c, 0dbe5, 0xe97e, 0xf8f7,
    0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
    0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
    0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
    0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
    0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
    0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
    0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
    0xce4c, 0xdfc5, 0xed5e, 0xfcfd, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
    0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
    0xdecd, 0xcf44, 0xfdd, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
    0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
    0xef4e, 0xfec7, 0xcc5c, 0xdd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
    0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
    0xffcf, 0xee46, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0xb70,
    0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
    0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
    0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
    0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
    0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
    0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
    0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
    0x39c3, 0x284a, 0x1ad1, 0xb58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
    0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
    0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
    0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
    0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
    0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
    0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
    0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
    0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};

#define PPPINITFCS16      0xffff /* Initial FCS value */
#define PPPGOODFCS16     0xf0b8 /* Good final FCS value */

/*
 * Calculate a new fcs given the current fcs and the new data.
 */

```

```

u16 pppfcs16(fcs, cp, len)
register u16 fcs;
register unsigned char *cp;
register int len;
{
    ASSERT(sizeof (u16) == 2);
    ASSERT((u16) -1 > 0);
    while (len--)
        fcs = (fcs >> 8) ^ fcstab[(fcs ^ *cp++) & 0xff];

    return (fcs);
}

/*
 * How to use the fcs
 */
tryfcs16(cp, len)
register unsigned char *cp;
register int len;
{
    u16 trialfcs;

    /* add on output */
    trialfcs = pppfcs16( PPPINITFCS16, cp, len );
    trialfcs ^= 0xffff;           /* complement */
    cp[len] = (trialfcs & 0x00ff);      /* least significant byte first */
    cp[len+1] = ((trialfcs >> 8) & 0x00ff);

    /* check on input */
    trialfcs = pppfcs16( PPPINITFCS16, cp, len + 2 );
    if ( trialfcs == PPPGOODFCS16 )
        printf("Good FCS\n");
}

```

### 8.5.4 FCS table generator

The following code creates the lookup table used to calculate the FCS-16.

```

/*
 * Generate a FCS-16 table.
 *
 * Drew D. Perkins at Carnegie Mellon University.
 *
 * Code liberally borrowed from Mohsen Banan and D. Hugh Redelmeier.
 */

/*
 * The FCS-16 generator polynomial: x**0 + x**5 + x**12 + x**16.
 */
#define P      0x8408

/*
 * NOTE The hex to "least significant bit" binary always causes
 * confusion, but it is used in all HDLC documents. Example: 0x03
 * translates to 1100 0000B. The above defined polynomial value
 * (0x8408) is required by the algorithm to produce the results
 * corresponding to the given generator polynomial
 * (x**0 + x**5 + x**12 + x**16)
 */

main()
{
    register unsigned int b, v;
    register int i;

    printf("typedef unsigned short u16;\n");
    printf("static u16 fcstab[256] = {");
    for (b = 0; ; ) {
        if (b % 8 == 0)
            printf("\n");

        v = b;
        for (i = 8; i--; )
            v = v & 1 ? (v >> 1) ^ P : v >> 1;

        printf("\t0x%04x", v & 0xFFFF);
        if (++b == 256)
            break;
        printf(",");
    }
    printf("\n};\n");
}

```

## 8.6 Data link layer management services

### 8.6.1 Overview

Figure 85 shows management services provided by the data link layer to the system management process. The same service set is used both at the client and the server sides. As these services are of local importance only, these clauses are included here only as guidance.

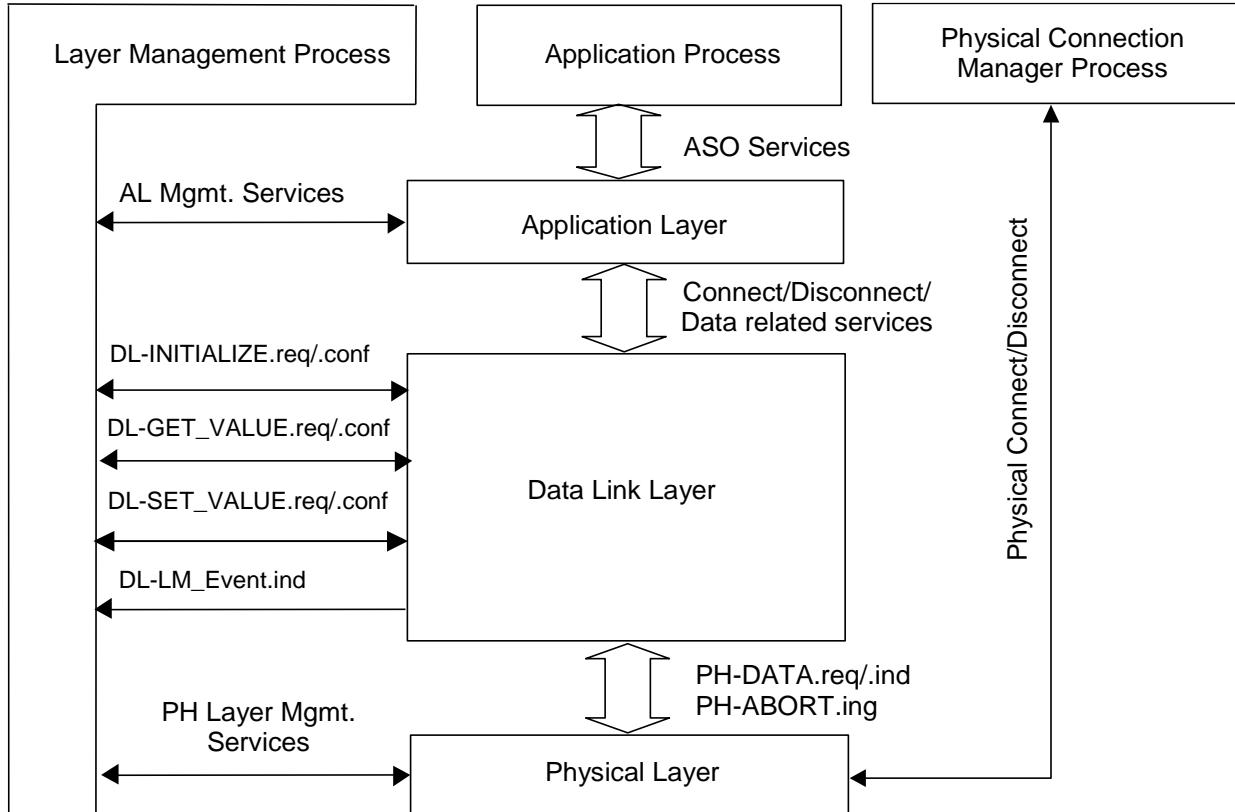


Figure 85 – Layer management services

### 8.6.2 Data link layer management service definitions

#### 8.6.2.1 DL-INITIALIZE.request

##### 8.6.2.1.1 Function

This primitive is the service request primitive of the DL-INITIALIZE service.

##### 8.6.2.1.2 Semantics of the service primitive

The semantics of this service primitive is as follows:

```
DL-INITIALIZE.request
(
)
```

##### 8.6.2.1.3 Use

The DL-INITIALIZE.request primitive is invoked by the layer management process to initialize data link layer parameters to their default values; see 8.4.5.6.

#### 8.6.2.2 DL-INITIALIZE.confirm

##### 8.6.2.2.1 Function

This primitive is the service confirm primitive of the DL-INITIALIZE service.

##### 8.6.2.2.2 Semantics of the service primitive

The semantics of this service primitive is as follows:

```
DL-INITIALIZE.confirm  (
    Result
)
```

The value of the Result parameter indicates, whether the data link layer parameters have been successfully initialized or not.

#### 8.6.2.2.3 Use

The DL-INITIALIZE.confirm primitive is generated by the data link layer to locally confirm the result of the preceding DL-INITIALIZE.request service invocation.

#### 8.6.2.3 DL-GET\_VALUE.request

##### 8.6.2.3.1 Function

This primitive is the service request primitive of the DL-GET\_VALUE service.

##### 8.6.2.3.2 Semantics of the service primitive

The semantics of this primitive is as follows:

```
DL-GET_VALUE.request (
    Layer_Parameter_Identifier_List
)
```

The Layer\_Parameter\_Identifier\_List parameter indicates the required layer parameters.

##### 8.6.2.3.3 Use

The DL-GET\_VALUE.request primitive is used by the layer management process to obtain the value of one or more layer parameters from the data link layer.

#### 8.6.2.4 DL-GET\_VALUE.confirm

##### 8.6.2.4.1 Function

This primitive is the service confirm primitive of the DL-GET\_VALUE service.

##### 8.6.2.4.2 Semantics of the service primitive

The semantics of this primitive is as follows:

```
DL-GET_VALUE.confirm (
    Layer_Parameter_GetResult_List
)
```

The Layer\_Parameter\_GetResult\_List parameter carries the identifier(s) of the required parameter(s); the result of the GET operation applied to this parameter and in the case of a successful operation the value of the required layer parameters.

##### 8.6.2.4.3 Use

The DL-GET\_VALUE.confirm primitive is generated by the data link layer to indicate the result of the invocation of the DL-GET\_VALUE.request primitive and to give back the layer parameters.

#### 8.6.2.5 DL-SET\_VALUE.request

##### 8.6.2.5.1 Function

This primitive is the service request primitive of the DL-SET\_VALUE service.

##### 8.6.2.5.2 Semantics of the service primitive

The semantics of this primitive is as follows:

```
DL-SET_VALUE.request (
    Layer_Parameter_Value_List
)
```

The Layer\_Parameter\_Value\_List parameter carries the identifier(s) and the value(s) of the required layer parameters to be set.

#### 8.6.2.5.3 Use

The DL-SET\_VALUE.request primitive is invoked by the layer management process to set the value of one or more layer parameters of the data link layer.

#### 8.6.2.6 DL-SET\_VALUE.confirm

##### 8.6.2.6.1 Function

This primitive is the service confirm primitive of the DL-SET\_VALUE service.

##### 8.6.2.6.2 Semantics of the service primitive

The semantics of this primitive is as follows:

```
DL-SET_VALUE.confirm  (
    Layer_Parameter_SetResult_List
)
```

The Layer\_Parameter\_SetResult\_List parameter carries the identifier(s) of the required parameter(s) and the result of the SET operation applied to this parameter.

##### 8.6.2.6.3 Use

The DL-SET\_VALUE.confirm primitive is generated by the data link layer to indicate the result of the invocation of the previous DL-SET\_VALUE.request primitive.

#### 8.6.2.7 DL-LM\_EVENT.indication

##### 8.6.2.7.1 Function

This primitive is the service indication primitive of the DL-LM\_EVENT service.

##### 8.6.2.7.2 Semantics of the service primitive

The semantics of this primitive is as follows:

```
DL-LM_EVENT.indication  (
    Event_Identifier,
    Event_Parameters
)
```

The Event\_Identifier parameter carries the identifier of the event(s) occurred.

The Event\_Parameters parameter, if present, may give some additional information.

##### 8.6.2.7.3 Use

The DL-LM\_EVENT.indication primitive is generated to indicate the occurrence of a data link layer event to the layer management process.

## 9 DLMS/COSEM application layer

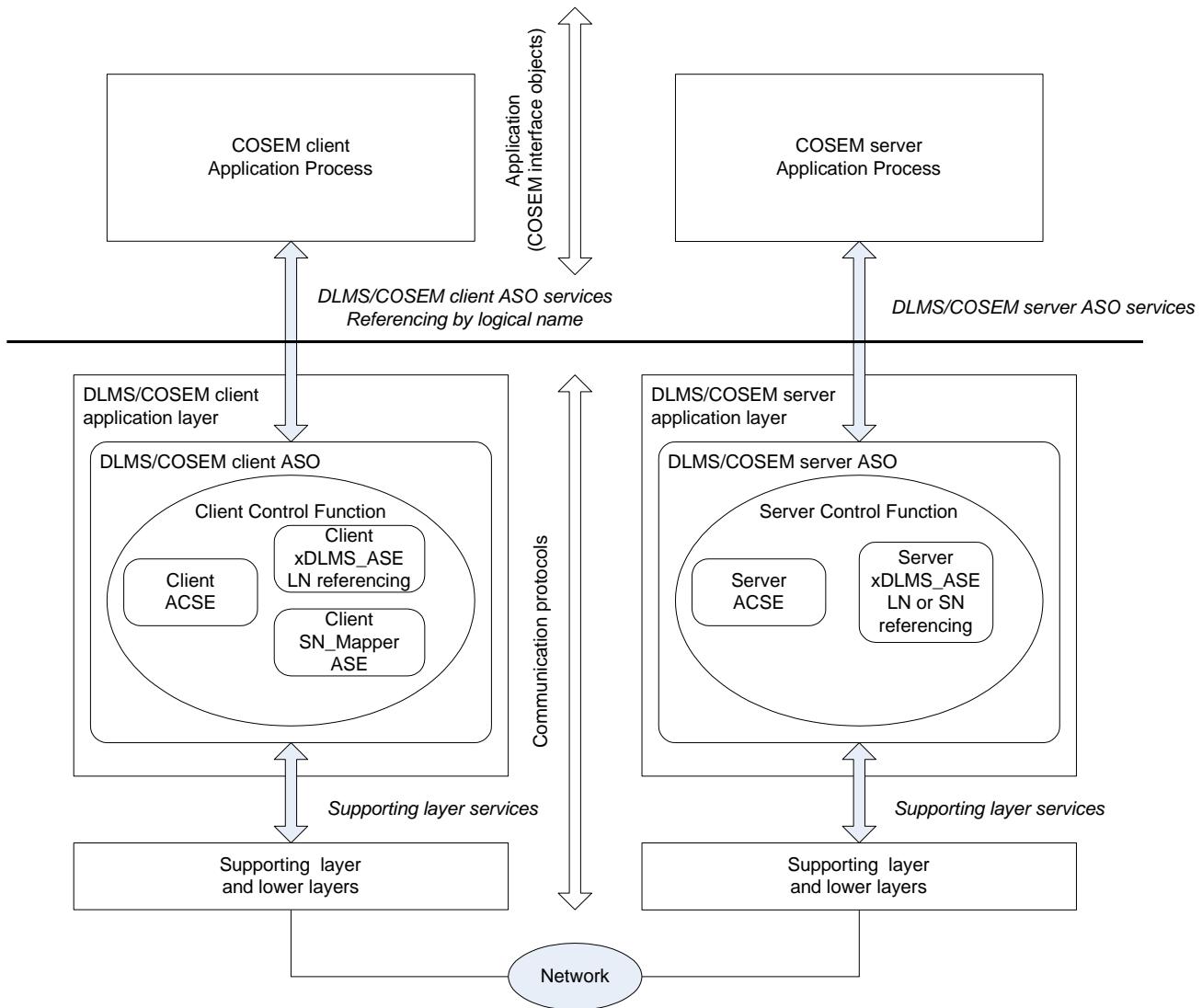
### 9.1 DLMS/COSEM application layer main features

#### 9.1.1 General

This subclause 9.1 provides an overview of the main features provided by the DLMS/COSEM AL.

#### 9.1.2 DLMS/COSEM application layer structure

The structure of the client and server DLMS/COSEM application layers is shown in Figure 86.



**Figure 86 – The structure of the DLMS/COSEM application layers**

The main component of the DLMS/COSEM AL is the Application Service Object (ASO). It provides services to its service user, the COSEM Application Process (APs) and uses services provided by the supporting protocol layer. It contains three mandatory components both on the client and on the server side:

- the Association Control Service Element, ACSE;
- the extended DLMS Application Service Element, xDLMS ASE;
- the Control Function, CF.

On the client side, there is a fourth, optional element, called the Client SN\_Mapper ASE.

The ACSE provides services to establish and release application associations (AAs). See 9.1.3.

The xDLMS ASE provides services to transport data between COSEM APs. See 9.1.4.

The Control Function (CF) element specifies how the ASO services invoke the appropriate service primitives of the ACSE, the xDLMS ASE and the services of the supporting protocol layer. See also 9.4.1.

Both the client and the server DLMS/COSEM ASO may contain other, optional application protocol components.

The optional Client SN\_Mapper ASE is present in the client side AL ASO, when the server uses SN referencing. It provides mapping between services using LN and SN referencing. See 9.1.5.

The DLMS/COSEM AL performs also some functions of the OSI presentation layer:

- encoding and decoding the ACSE APDUs and the xDLMS APDUs, see also 9.4.3;
- alternatively, generating and using XML documents representing ACSE and xDLMS APDUs;
- applying compression and decompression;
- applying, verifying and removing cryptographic protection.

### 9.1.3 The Association Control Service Element, ACSE

For the purposes of DLMS/COSEM connection oriented (CO) communication profiles, the CO ACSE, specified in ISO/IEC 15953:1999 and ISO/IEC 15954:1999 is used.

The services provided for application association establishment and release are the following:

- COSEM-OPEN;
- COSEM-RELEASE;
- COSEM-ABORT.

The COSEM-OPEN service is used to establish AAs. It is based on the ACSE A-ASSOCIATE service. It causes the start of use of an AA by those ASE procedures identified by the value of the Application\_Context\_Name, Security\_Mechanism\_Name and xDLMS context parameters. AAs may be established in different ways:

- confirmed AAs are established via a message exchange – using the COSEM-OPEN service – between the client and the server to negotiate the contexts. Confirmed AAs can be established between a single client and a single server;
- unconfirmed AAs are established via a message sent – using the COSEM-OPEN service – from the client to the server, using the parameters of the contexts supposed to be supported by the server. Unconfirmed AAs can be established between a client and one or multiple servers;
- pre-established AAs may pre-exist. In this case, the COSEM-OPEN service is not used. The client has to be aware of the contexts supported by the server. A pre-established AA can be confirmed or unconfirmed.

The COSEM-RELEASE service is used to release AAs. If successful, it causes the completion of the use of the AA without loss of information in transit (graceful release). In some communication profiles – for example in the TCP-UDP/IP based profile – the COSEM-RELEASE service is based on the ACSE A-RELEASE service. In some other communication profiles – for example in the 3-layer, CO, HDLC based profile – there is a one-to-one relationship between a confirmed AA and the supporting protocol layer connection. In such profiles AAs can be released simply by disconnecting the corresponding supporting protocol layer connection. Pre-established AAs cannot be released.

The COSEM-ABORT service causes the abnormal release of an AA with the possible loss of information in transit. It does not rely on the ACSE A-ABORT service.

The COSEM-OPEN service is specified in 9.3.2, the COSEM-RELEASE service in 9.3.3 and the COSEM-ABORT service in 9.3.4.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	195/614
-----------------------	------------	-----------------------	---------

### 9.1.4 The xDLMS application service element

#### 9.1.4.1 Overview

To access attributes and methods of COSEM objects, the services of the xDLMS ASE are used. It is based on the DLMS standard, IEC 61334-4-41:1996. This Technical Report specifies a range of extensions to extend functionality while maintaining backward compatibility. The extensions comprise the following:

- additional services, see 9.1.4.3;
- additional mechanisms, see 9.1.4.4;
- additional data types, see 9.1.4.5;
- new DLMS version number, see 9.1.4.6;
- new conformance block, see 9.1.4.7;
- clarification of the meaning of the PDU size, see 9.1.4.8.

#### 9.1.4.2 The xDLMS initiate service

To establish the xDLMS context the xDLMS Initiate service — specified in IEC 61334-4-41:1996, 5.2 — is used. This service is integrated in the COSEM-OPEN service, see 9.3.2.

#### 9.1.4.3 COSEM object related xDLMS services

##### 9.1.4.3.1 General

COSEM object related xDLMS services are used to access COSEM object attributes and methods.

DLMS UA 1000-1 Part 2 Ed.15:2021, 4.1.2 specifies two referencing methods:

- Logical Name (LN) referencing; and
- Short Name (SN) referencing.

For more information on referencing methods, see 9.1.4.4.2.

Therefore, two distinct xDLMS service sets are specified: one exclusively using Logical Name (LN) referencing and the other exclusively using short name (SN) referencing. It can be considered that there are two different xDLMS ASEs: one providing services with LN referencing and the other with SN referencing. The client ASO always uses the xDLMS ASE with LN referencing. The server ASO may use either the xDLMS ASE with LN referencing or the xDLMS ASE with SN referencing or both.

These services may be:

- requested / solicited by the client; or
- unsolicited: these are always initiated by the server without a previous request from the client.

Services requested by the client may be also (see 9.4.6.2):

- confirmed: in this case, the server provides a response to the request;
- unconfirmed: in this case, the server does not provide a response to the request.

Unsolicited DataNotification from the server may be also (see 9.3.10):

- confirmed: in this case, the client provides a response to acknowledge the receipt of the unsolicited DataNotification
- unconfirmed: in this case, the client does not provide a response to the unsolicited DataNotification.

The additional services – which are not based on DLMS services specified in IEC 61334-4-41:1996 – are:

- the GET, SET, ACTION and ACCESS used to access COSEM object attributes and methods using LN referencing;
- the DataNotification service used by the server to push data to the client;
- the EventNotification service used by the server to notify the client about events that occur in the server.

196/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

#### 9.1.4.3.2 xDLMS services used by the client with LN referencing

In the case of LN referencing, COSEM object attributes and methods are referenced via the identifier of the COSEM object instance to which they belong. For this referencing method, the following additional services are specified:

- the GET service is used by the client to request the server to return the value of one or more attributes, see 9.3.6;
- the SET service is used by the client to request the server to replace the content of one or more attributes, see 9.3.7;
- the ACTION service is used by the client to request the server to invoke one or more methods. Invoking methods may imply sending method invocation parameters and receiving return parameters, see 9.3.8;
- the ACCESS service, a unified service which can be used by the client to access multiple attributes and/or methods with a single .request; see 9.3.9.

These services can be invoked by the client in a confirmed or unconfirmed manner.

#### 9.1.4.3.3 xDLMS services used by the client with SN referencing

In the case of SN referencing, COSEM object attributes and methods are mapped to DLMS named variables specified in IEC 61334-4-41:1996, 10.1.2.

The xDLMS services using SN referencing are based on the DLMS variable access services, specified in IEC 61334-4-41:1996 subclauses 10.4 – 10.6 and they are the following:

- the Read service is used by the client to request the server to return the value of one or more attributes or to invoke one or more methods when return parameters are expected. It is a confirmed service. See 9.3.14;
- the Write service is used by the client to request the server to replace the content of one or more attributes or to invoke one or more methods when no return parameters are expected. It is a confirmed service. See 9.3.15;
- the UnconfirmedWrite service is used by the client to request the server to replace the content of one or more attributes or to invoke one or more methods when no return parameters are expected. It is an unconfirmed service. See 9.3.16.

New variants of the Variable\_Access\_Specification service parameter (see 9.3.13), the Read.response and the Write.response services have been added to support selective access – see 9.1.4.3.5 – and block transfer, see 9.1.4.4.5.

#### 9.1.4.3.4 Unsolicited services

Unsolicited services are initiated by the server, on pre-defined conditions, e.g. schedules, triggers or events, to inform the client of the value of one or more attributes, as though they had been requested by the client.

To support push operation, the DataNotification service is available, see 9.3.10. It can be used in application contexts using either SN referencing or LN referencing.

**NOTE** The DataNotification service is used in conjunction with “Push setup” COSEM objects, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.8.

To support event notification, the following unsolicited services are available:

- with LN referencing, the EventNotification service, see 9.3.11;
- with SN referencing, the InformationReport service, see 9.3.17. This service is based on IEC 61334-4-41:1996, 10.7.

#### 9.1.4.3.5 Selective access

In the case of some COSEM interface classes, selective access to attributes is available, meaning that either the whole attribute or a selected portion of it can be accessed as required. For this purpose, access selectors and parameters are specified as part of the specification of the relevant attributes.

To use this possibility, attribute-related services can be invoked with access selection parameters. In the case of LN referencing, this feature is called Selective access; see 9.3.6 and 9.3.7. It is a negotiable feature; see 9.4.6.1. In the case of SN referencing, this feature is called Parameterized access; see 9.3.14, 9.3.15 and 9.3.16. It is a negotiable feature; see 9.4.6.1.

#### 9.1.4.3.6 Multiple references

In a COSEM object related service invocation, it is possible to reference one or several named variables, attributes and/or methods. Using multiple references is a negotiable feature. See 9.4.6.1.

#### 9.1.4.3.7 Attribute\_0 referencing

With the GET, SET and ACCESS services a special feature, Attribute\_0 referencing is available. By convention, attributes of COSEM objects are numbered from 1 to n, where Attribute\_1 is the logical name of the COSEM object. Attribute\_0 has a special meaning: it references all attributes with positive index (public attributes). The use of Attribute\_0 referencing with the GET service is explained in 9.3.6, with the SET service in 9.3.7, and with the ACCESS service in 9.3.9.

NOTE As specified in DLMS UA 1000-1 Part 2 Ed.15:2021, 4.1.2, manufacturers may add proprietary methods and/or attributes to any object, using negative numbers.

Attribute\_0 referencing is a negotiable feature, see 9.4.6.1.

### 9.1.4.4 Additional mechanisms

#### 9.1.4.4.1 Overview

xDLMS specifies several new mechanisms – compared to DLMS as specified in IEC 61334-4-41:1996 – to improve functionality, flexibility and efficiency. The additional mechanisms are:

- referencing using logical names;
- identification of service invocations;
- priority handling;
- transferring long application messages;
- composable xDLMS messages;
- compression and decompression;
- general cryptographic protection;
- general block transfer.

#### 9.1.4.4.2 Referencing methods and service mapping

To access COSEM object attributes and methods with the xDLMS services, they have to be referenced. As already mentioned in 9.1.4.3.1, DLMS UA 1000-1 Part 2 Ed.15:2021, 4.1.2 specifies two referencing methods:

- Logical Name (LN) referencing; and
- Short Name (SN) referencing.

In the case of LN referencing, COSEM object attributes and methods are referenced via the logical name (COSEM\_Object\_Instance\_ID) of the COSEM object instance to which they belong. In the case of SN referencing, COSEM object attributes and methods are mapped to DLMS named variables.

Accordingly, there are two xDLMS ASEs specified: one using xDLMS services with LN referencing and one using xDLMS services with SN referencing.

On the client side, in order to handle the different referencing methods transparently for the AP, the AL uses the xDLMS ASE with LN referencing. Using a unique, standardized service set between COSEM client APs and the communication protocol – hiding the particularities of DLMS servers using different referencing methods – allows specifying an Application Programming Interface, API. This is an explicitly specified interface corresponding to this service set for applications running in a given computing environment (for example Windows, UNIX, etc.) Using this – public – API specification, client applications can be developed without knowledge about particularities of a given server.

On the server side, either the xDLMS ASE with LN referencing or the xDLMS ASE with SN referencing or both xDLMS ASEs can be used.

198/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

In the case of confirmed AAs, the referencing method is negotiated during the AA establishment phase via the COSEM application context. It shall not change during the lifetime of the AA established. Using LN or SN services within a given AA is exclusive.

In the case of unconfirmed and pre-established AAs, the client AL is expected to know the referencing method supported by the server.

When the server uses LN referencing, the services are the same on both sides. When the server uses SN referencing the Client SN\_Mapper ASE in the client maps the SN referencing into LN referencing or vice versa. See 9.1.2 and 9.1.5.

#### **9.1.4.4.3 Identification of service invocations: the Invoke\_Id parameter**

In the client/server model, requests are sent by the client and responses are sent by the server. The client is allowed to send several requests before receiving the response to the previous ones, provided that this is allowed by the lower layers.

Therefore – to be able to identify which response corresponds to each request – it is necessary to include a reference in the request.

The Invoke\_Id parameter is used for this purpose. The value of this parameter is assigned by the client so that each request carries a different Invoke\_Id. The server shall copy the Invoke\_Id into the corresponding response.

In the ACCESS and the DataNotification service – see 9.3.9 and 9.3.10 – the Long-Invoke-Id parameter is used instead of the Invoke\_Id parameter.

The EventNotification service does not contain the Invoke\_Id parameter.

This feature is available only with LN referencing.

#### **9.1.4.4.4 Priority handling**

For data transfer services using LN referencing, two priority levels are available: normal (FALSE) and high (TRUE). This feature allows receiving a response to a new request before the response to a previous request is completed.

Normally, the server serves incoming service requests in the order of reception (FIFS, First In, First Served). However, a request with the priority parameter set to high (TRUE) is served before the previous requests with priority set to normal (FALSE). The response carries the same priority flag as that of the corresponding request. Managing priority is a negotiable feature; see 9.4.6.1.

NOTE 1 As service invocations are identified with an Invoke\_Id, services with the same priority can be served in any order.

NOTE 2 If the feature is not supported, requests with HIGH priority are served with NORMAL priority.

This feature is not available with services using SN referencing. The server treats the services on a FIFS basis.

#### **9.1.4.4.5 Transferring long messages**

The xDLMS service primitives are carried in an encoded form by xDLMS APDUs. This encoded form may be longer than the Client / Server Max Receive PDU Size negotiated. To transfer such 'long' messages, there are two mechanisms available:

- the general block transfer (GBT) mechanism specified in 9.1.4.4.9;
- the service-specific block transfer mechanism.

Using the general or the service-specific block transfer mechanism is a negotiable feature; see 9.4.6.1.

An APDU that fits in the Client / Server Max Receive PDU Size negotiated may be too long to fit in a single frame / packet of the supporting protocol layer. Such APDUs may be transported if the supporting protocol layer provide(s) segmentation; see Clause 10.

Service-specific block transfer mechanism is available with:

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	199/614
-----------------------	------------	-----------------------	---------

- confirmed services using LN referencing: GET, SET, ACTION;
- confirmed services using SN referencing: Read, Write.

and the related APDUs.

Service-specific block transfer is not available with:

- unconfirmed services;
- unsolicited services (DataNotification, EventNotification and InformationReport);
- the ACCESS service

and the related APDUs.

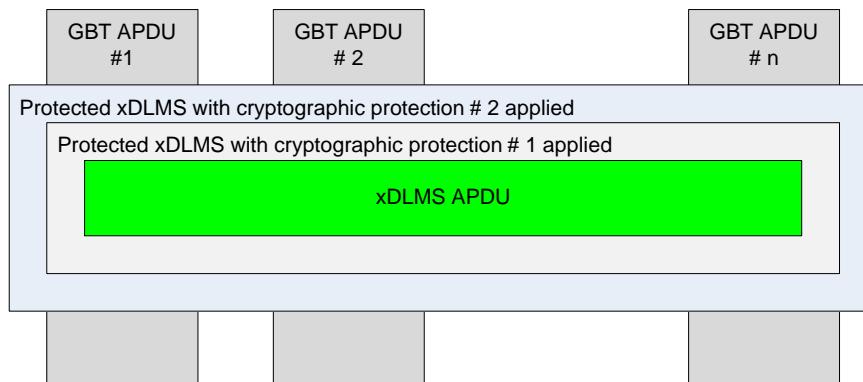
With service-specific block transfer, the reception of each block has to be acknowledged before the next block can be sent. A recovery mechanism for lost blocks is not available. When global key or dedicated key ciphering is required, it is applied to the APDU carrying a block – or an acknowledgement of a block – of the long message. This creates considerable computing and transport overhead. Service specific digital signature is not available.

Conversely, the GBT mechanism can be used with any xDLMS APDU including the general-ciphering and general-signing APDUs. It provides bidirectional block transfer, streaming and lost block recovery. When cryptographic protection is required, it is applied to the complete APDU and then the protected APDU is transferred in blocks, as specified in 9.1.4.4.6 Figure 87.

#### 9.1.4.4.6 Composable xDLMS messages

The three important aspects of dealing with xDLMS messages are encoding / decoding, applying, verifying / removing cryptographic protection and block transfer.

The concept of composable xDLMS messages separates the three aspects, as shown in Figure 87. See also Figure 113.



**Figure 87 – The concept of composable xDLMS messages**

Once the APDU corresponding to the service primitive invoked by the AP is built by the AL, the general protection mechanism can be used to apply cryptographic protection. When an unprotected or a protected APDU is too long to fit in the negotiated APDU size, then the general block transfer mechanism can be applied.

These mechanisms can be applied with all xDLMS APDUs.

NOTE 1 With the GET, SET, ACTION, EventNotification, Read and Write, UnconfirmedWrite and InformationReport services, service-specific cryptographic protection is available using specific service protection types and APDUs.

NOTE 2 With the GET, SET, ACTION, Read, Write, and UnconfirmedWrite and services, service-specific block transfer is available using specific service request / response types and APDUs.

#### 9.1.4.4.7 Compression and decompression

In order to optimize the use of communication media, it is possible to compress xDLMS APDUs to be sent and decompress xDLMS APDUs received. For details, see 9.2.3.6.

#### 9.1.4.4.8 General protection

This mechanism can be used to apply cryptographic protection to any xDLMS APDU and this allows applying multiple layers of protection between the client and the server or between a third party and the server. See also 9.2.2.5.

For this purpose, the following APDUs are available; see 9.2.7.2.3:

- the general-ded-ciphering and the general-glo-ciphering APDUs;
- the general-ciphering APDUs;
- the general-signing APDU.

Using the general protection mechanism is a negotiable feature, see 9.4.6.1.

#### 9.1.4.4.9 General block transfer (GBT)

The GBT mechanism can be used to transfer any – short or long – xDLMS APDU in blocks. With GBT, the blocks are carried by general-block-transfer APDUs instead of service-specific “with-datablock” APDUs.

The GBT mechanism supports bi-directional block transfer, streaming and lost block recovery:

- bi-directional block transfer means that while one party is sending blocks, the other party not only confirms the blocks received but if it has blocks to send it can send them as well while it is still receiving blocks;  
NOTE Bi-directional block transfer is useful when long service parameters need to be transported in both directions.
- streaming means that several blocks may be sent – streamed – by one party without an acknowledgement of each block from the other party;
- lost block recovery means that if the reception of a block sent is not confirmed, it can be sent again. If streaming is used, lost block recovery takes place at the end of each streaming window.

The GBT mechanism is managed by the AL using the block transfer streaming parameters specified in 9.3.5.

The protocol of the general block transfer mechanism is specified in 9.4.6.13.

The GBT mechanism supports the following use cases:

- 1) short confirmed request by the client that leads to a long response from the server: the request may be sent with or without using GBT. If it is sent using GBT then the client may advertise its preferred GBT window size when the exchange is started. The server responds using GBT;
- 2) long confirmed request by the client: the client sends the request using GBT and the server sends the response using GBT, whatever is the length of the response;
- 3) long unconfirmed request by the client: the client sends the request using GBT;
- 4) long unsolicited confirmed request by the server: the server sends the request and the client sends the response using GBT;
- 5) long unsolicited unconfirmed request by the server: the server sends the request using GBT.

#### 9.1.4.5 Additional data types

The additional data types are specified in 9.5 and in 9.6.

#### 9.1.4.6 xDLMS version number

The new DLMS version number, corresponding to the first version of the xDLMS ASE is 6.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	201/614
-----------------------	------------	-----------------------	---------

#### 9.1.4.7 xDLMS conformance block

The xDLMS conformance block enables optimised DLMS server implementations with extended functionality. It can be distinguished from the DLMS conformance block by its tag "Application 31". See 9.4.6.1, 9.5 and in 9.6.

The xDLMS conformance block is part of the xDLMS context.

In the case of confirmed AAs, the conformance block is negotiated during the AA establishment phase via the xDLMS context. It shall not change during the lifetime of the AA established.

In the case of unconfirmed and pre-established AAs, the client AL is expected to know the conformance block supported by the server.

#### 9.1.4.8 Maximum PDU size

To clarify the meaning of the maximum PDU size usable by the client and the server, the modifications shown in Table 20 have been made. The xDLMS Initiate service uses these names for PDU sizes.

**Table 20 – Clarification of the meaning of PDU size for DLMS/COSEM**

was:	new:
<b>IEC 61334-4-41:1996, 5.2.2, Table 3</b>	
Proposed Max PDU Size	Client Max Receive PDU Size
Negotiated Max PDU Size	Server Max Receive PDU Size
<b>IEC 61334-4-41:1996, 5.2.3, 7<sup>th</sup> paragraph</b>	
The Proposed Max PDU Size parameter, of type Unsigned16, proposes a maximum length expressed in bytes for the exchanged DLMS APDUs. The value proposed in an Initiate request must be large enough to always permit the Initiate Error PDU transmission	The Client Max Receive PDU Size parameter, of type Unsigned16, contains the maximum length expressed in bytes for a DLMS APDU that the server may send. The client will discard any received PDUs that are longer than this maximum length. The value must be large enough to always permit the AARE APDU transmission. Values below 12 are reserved. The value 0 indicates that there is no limit on the PDU size.
<b>IEC 61334-4-41:1996, 5.2.3, last paragraph</b>	
The Negotiated Max PDU Size parameter, of type Unsigned16, contains a maximum length expressed in bytes for the exchanged DLMS APDUs. A PDU that is longer than this maximum length will be discarded. This maximum length is computed as the minimum of the Proposed Max PDU Size and the maximum PDU size than the VDE-handler may support.	The Server Max Receive PDU Size parameter, of type Unsigned16, contains the maximum length expressed in bytes for a DLMS APDU that the client may send. The server will discard any received PDUs that are longer than this maximum length. Values below 12 are reserved. The value 0 indicates that there is no limit on the PDU size.

#### 9.1.5 Layer management services

Layer management services have local importance only. Therefore, specification of these services is not within the Scope of this Technical Report.

The specific SetMapperTable service is defined in 9.3.18.

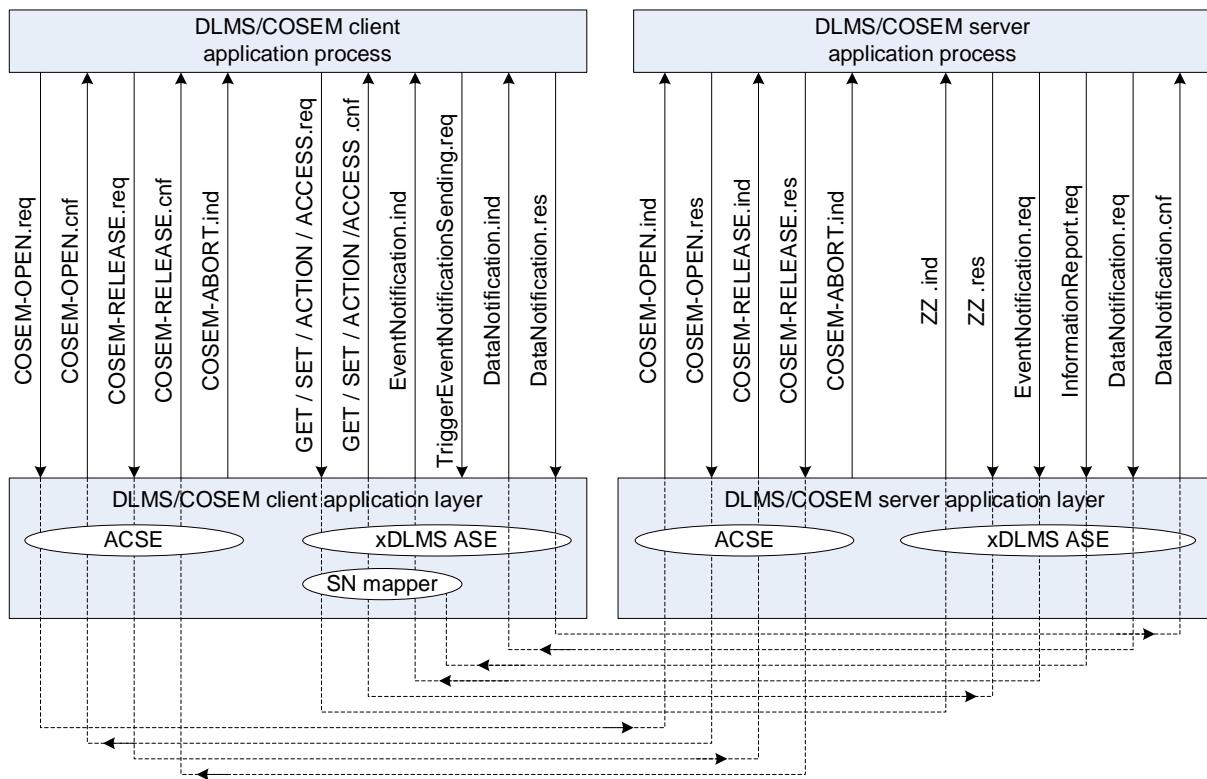
#### 9.1.6 Summary of DLMS/COSEM application layer services

A summary of the services available at the top of the DLMS/COSEM AL is shown in Figure 88. Layer management services are not shown. Although the service primitives are different on the client and server side, the APDUs are the same.

NOTE 1 For example, when the client AP invokes a GET.request service primitive the client AL builds a GET-Request APDU. When this is received by the server AL, it invokes a GET.ind service primitive.

The DLMS/COSEM AL services are specified in 9.3. The DLMS/COSEM AL protocol is specified in 9.4. The abstract syntax of the ACSE and xDLMS APDUs is specified in 9.5. The XML schema is defined in 9.6.

Encoding examples are provided in Clauses 11, 12, 13 and 14.



NOTE 2 The client AP always uses LN referencing. If the server uses SN referencing then a mapping is performed by the Client SN\_Mapper ASE. Consequently, the service primitives ZZ.ind and ZZ.res may be LN or SN service primitives. LN/SN service mapping is specified in 9.5.

NOTE 3 The ACCESS service cannot be mapped to services using SN referencing.

**Figure 88 – Summary of DLMS/COSEM AL services**

#### 9.1.7 DLMS/COSEM application layer protocols

The DLMS/COSEM AL protocols specify the procedures for information transfer for AA control and authentication using connection-oriented ACSE procedures, and for data transfer between COSEM clients and servers using xDLMS procedures. Therefore, the DLMS/COSEM AL protocol is based on the ACSE standard as specified in ISO/IEC 15954:1999 and the DLMS standard, as specified in IEC 61334-4-41:1996, with the extensions for DLMS/COSEM. The procedures are defined in terms of:

- the interactions between peer ACSE and xDLMS protocol machines through the use of services of the supporting protocol layer;
- the interactions between the ACSE and xDLMS protocol machines and their service user.

The DLMS/COSEM AL protocols are specified in 9.4.

## 9.2 Information security in DLMS/COSEM

### 9.2.1 Overview

This subclause 9.2 describes and specifies:

- the DLMS/COSEM security concept, see 9.2.2;
- the cryptographic algorithms selected, see 9.2.3;
- the security keys, see 9.2.4, 9.2.5 and 9.2.6;
- the use of the cryptographic algorithms for entity authentication, xDLMS APDU protection and COSEM data protection, see 9.2.7.

### 9.2.2 The DLMS/COSEM security concept

#### 9.2.2.1 Overview

The resources of DLMS servers – COSEM object attributes and methods – can be accessed by DLMS clients within Application Associations, see also 4.5.

During an AA establishment the client and the server have to identify themselves. The server may also require that the *user* of a client identifies itself. Furthermore, the server may require that the client authenticates itself and the client may also require that the server authenticates itself. The identification and authentication mechanisms are specified in 9.2.2.2.

Once an AA is established, xDLMS services can be used to access COSEM object attributes and methods, subject to the security context and access rights. See 9.2.2.3 and 9.2.2.4.

The xDLMS APDUs carrying the services primitives can be cryptographically protected. The required protection is determined by the security context and the access rights. To support end-to-end security between third parties and servers, such third parties can also access the resources of a server using a client as a broker. The concept of message protection is further explained in 9.2.2.5.

Moreover, COSEM data carried by the xDLMS APDUs can be cryptographically protected; see 9.2.2.6.

As these security mechanisms are applied on the application process / application layer level, they can be used in all DLMS/COSEM communication profiles.

NOTE Lower layers may provide additional security.

#### 9.2.2.2 Identification and authentication

##### 9.2.2.2.1 Identification

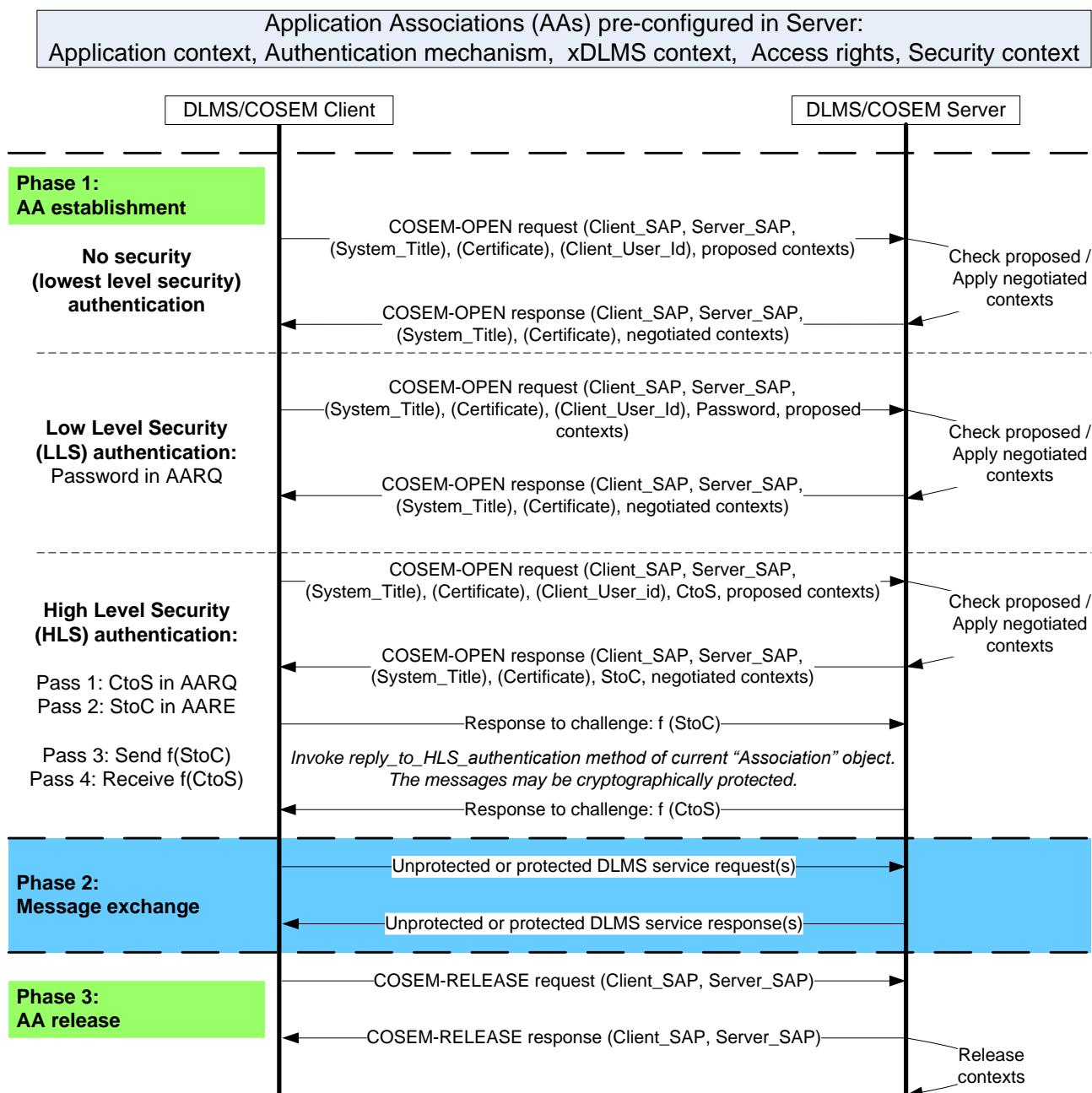
As specified in 4.3.3, DLMS/COSEM AEs are bound to Service Access Points (SAPs) in the protocol layer supporting the AL. These SAPs are present in the PDUs carrying the xDLMS APDUs within an AA.

The client user identification mechanism enables the server to distinguish between different users on the client side — that may be operators or third parties — to log their activities accessing the device. See also 4.3.6.

##### 9.2.2.2.2 Authentication mechanisms

###### 9.2.2.2.2.1 Overview

The authentication mechanisms determine the protocol to be used by the communication entities to authenticate themselves during AA establishment.



NOTE 1 The COSEM-OPEN service primitives are carried by AARQ / AARE APDUs. The COSEM-RELEASE service primitives are carried by RLRQ / RLRE APDUs (when used). See 9.3.2 and 9.3.3.

NOTE 2 The elements (System\_Title), (Certificate) and (Client\_User\_Id) are optional.

NOTE 3 In pre-established AAs no authentication takes place.

NOTE 4 The COSEM-RELEASE service can be cryptographically protected by including a ciphered xDLMS Initiate .request / .response APDU in the RLRQ.

**Figure 89 – Authentication mechanisms**

There are three different authentication mechanisms available with different authentication security levels:

- no security (Lowest Level Security) authentication; see 9.2.2.2.2;
- Low Level Security (LLS) authentication, see 9.2.2.2.3;

NOTE 1 In ITU-T X.811:1995 this is known as unilateral authentication, class 0 mechanism.

- High Level Security (HLS) authentication, see 9.2.2.2.4.

NOTE 2 In ITU-T X.811:1995 this is known as mutual authentication using challenge mechanisms.

They are shown in Figure 89. Authentication mechanisms are identified by names, see 9.4.2.2.3

The security of the message exchange (in Phase 2) is independent of the client-server authentication during AA establishment (Phase 1). Even in the case where no client-server authentication takes place, cryptographically protected APDUs can be used to ensure message security.

#### **9.2.2.2.2 No security (Lowest Level Security) authentication**

The purpose of No security (Lowest Level Security) authentication is to allow the client to retrieve some basic information from the server. This authentication mechanism does not require any authentication; the client can access the COSEM object attributes and methods within the security context and access rights prevailing in the given AA.

#### **9.2.2.2.3 Low Level Security (LLS) authentication**

In this case, the server requires that the client authenticates itself by supplying a password that is known by the server. The password is held by the current “Association SN / LN” object modelling the AA to be established. The “Association SN / LN” objects provide means to change the secret.

If the password supplied is accepted, the AA can be established, otherwise it shall be rejected.

LLS authentication is supported by the COSEM-OPEN service – see 9.3.2 – as follows:

- the client transmits a “secret” (a password) to the server, using the COSEM-OPEN.request service primitive;
- the server checks if the “secret” is correct;
- if yes, the client is authenticated and the AA can be established. From this moment, the negotiated contexts are valid;
- if not, the AA shall be rejected;
- the result of establishing the AA shall be sent back by the server using the COSEM-OPEN.response service primitive, together with diagnostic information.

#### **9.2.2.2.4 High Level Security (HLS) authentication**

In this case, both the client and the server have to successfully authenticate themselves to establish an AA. HLS authentication is a four-pass process that is supported by the COSEM-OPEN service and the *reply\_to\_HLS\_authentication* method of the “Association SN / LN” interface class:

- Pass 1: The client transmits a “challenge” *CtoS* and – depending on the authentication mechanism – additional information to the server;
- Pass 2: The server transmits a “challenge” *StoC* and – depending on the authentication mechanism – additional information to the client;

If *StoC* is the same as *CtoS*, the client shall reject it and shall abort the AA establishment process.

- Pass 3: The client processes *StoC* and the additional information according to the rules of the HLS authentication mechanism valid for the given AA and sends the result to the server. The server checks if *f(StoC)* is the result of correct processing and – if so – it accepts the authentication of the client;
- Pass 4: The server processes then *CtoS* and the additional information according to the rules of the HLS authentication mechanism valid for the given AA and sends the result to the client. The client checks if *f(CtoS)* is the result of correct processing and – if so – it accepts the authentication of the server.

Pass 1 and Pass 2 are supported by the COSEM-OPEN service.

After Pass 2 – provided that the proposed application context and xDLMS context are acceptable – the server grants access to the method `reply_to_HLS_authentication` of the current "Association SN / LN" object using the application context negotiated.

Pass 3 and Pass 4 are supported by the method `reply_to_HLS_authentication` of the "Association SN / LN" object(s). If both passes 3 and 4 are successfully executed, then the AA is established with the application context and xDLMS context negotiated.

The dedicated-key, if transferred, can be used from this moment.

Otherwise, either the client or the server aborts.

There are several HLS authentication mechanisms available. These are further specified in 9.2.7.4.

In some HLS authentication mechanisms, the processing of the challenges involves the use of an HLS secret.

The "Association SN / LN" interface class provides a method to change the HLS "secret": `change_HLS_secret`.

**REMARK** After the client has issued the `change_HLS_secret()` – or `change_LLS_secret()` – method, it expects a response from the server acknowledging that the secret has been changed. It is possible that the server transmits the acknowledgement, but due to communication problems, the acknowledgement is not received at the client side. Therefore, the client does not know if the secret has been changed or not. For simplicity reasons, the server does not offer any special support for this case; i.e. it is left to the client to cope with this situation.

### 9.2.2.3 Security context

The security context defines security attributes relevant for cryptographic transformations and includes the following elements:

- the security suite, determining the security algorithms available, see 9.2.3.7;
- the security policy, determining the kind(s) of protection to be applied generally to all xDLMS APDUs exchanged within an AA. The possible security policies are specified in 9.2.7.2.2;
- the security material, relevant for the given security algorithms, that includes security keys, initialization vectors, public key certificates and the like. As the security material is specific for each security algorithm, the elements are specified in detail in the relevant clauses.

The security context is managed by "Security setup" objects; see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7.

### 9.2.2.4 Access rights

Access rights to attributes may be: `no_access`, `read_only`, `write_only`, or `read_and_write`. Access rights to methods may be `no_access` or `access`.

In addition, access rights may stipulate cryptographic protection to be applied to xDLMS APDUs carrying the service primitives used to access a particular COSEM object attribute / method. The protection required on the .request and on the .response can be independently configured.

Access rights are held by the relevant "Association SN / LN" objects; see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.3 and 4.4.4. The possible access rights are specified in 9.2.7.2.2.

The protection to be applied shall meet the stronger of the requirement stipulated by the security policy and the access rights.

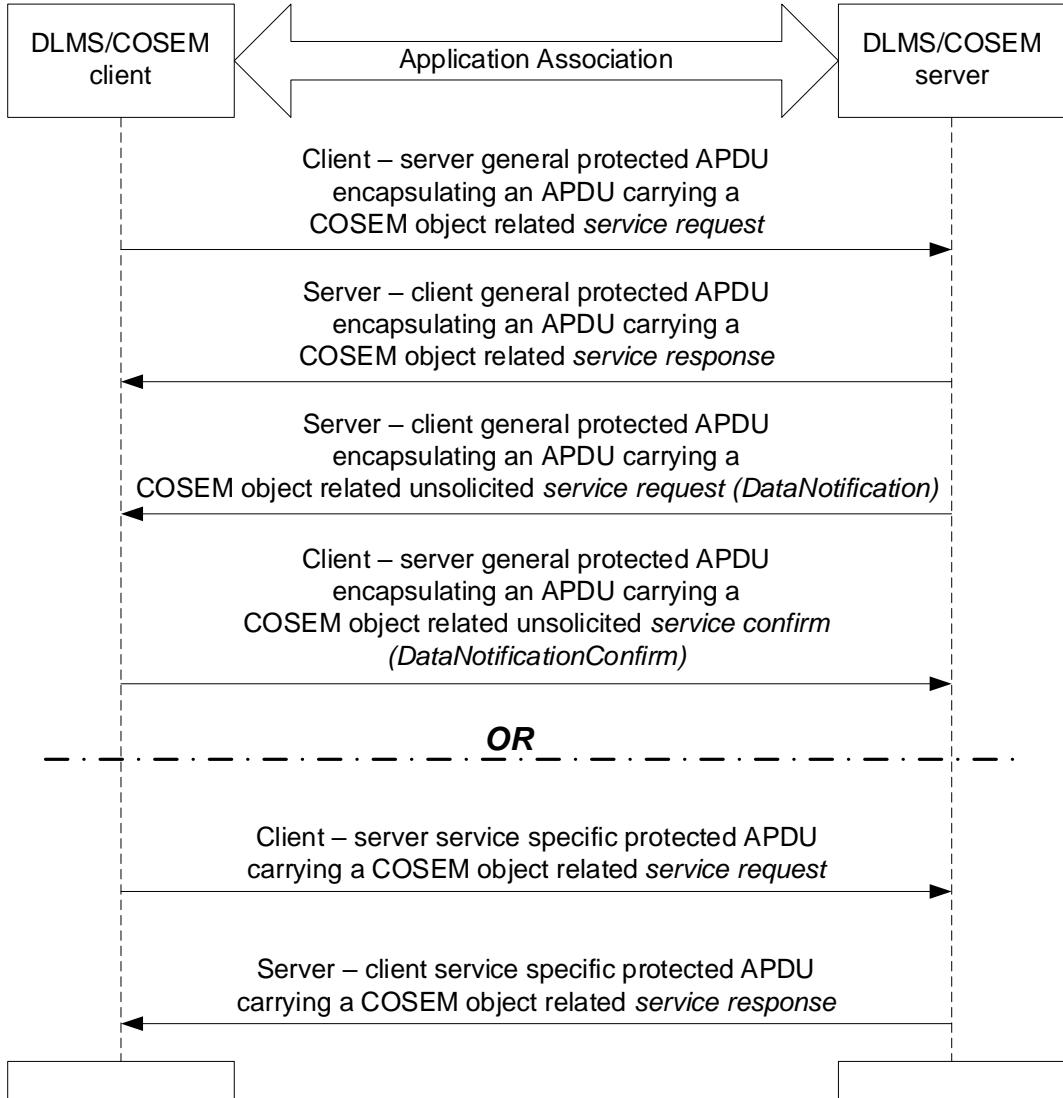
### 9.2.2.5 Application layer message security

DLMS/COSEM ensures AL level security by providing means to cryptographically protect xDLMS APDUs. The protection may be any combination of authentication, encryption and digital signature and can be applied in a multi-layer fashion by multiple parties. The protection is applied by the originator and is verified and removed by the recipient.

A request or response received shall be processed only if the protection on the message carrying the request or response could be successfully verified and removed.

Project specific companion specifications may specify additional criteria for accepting and processing messages.

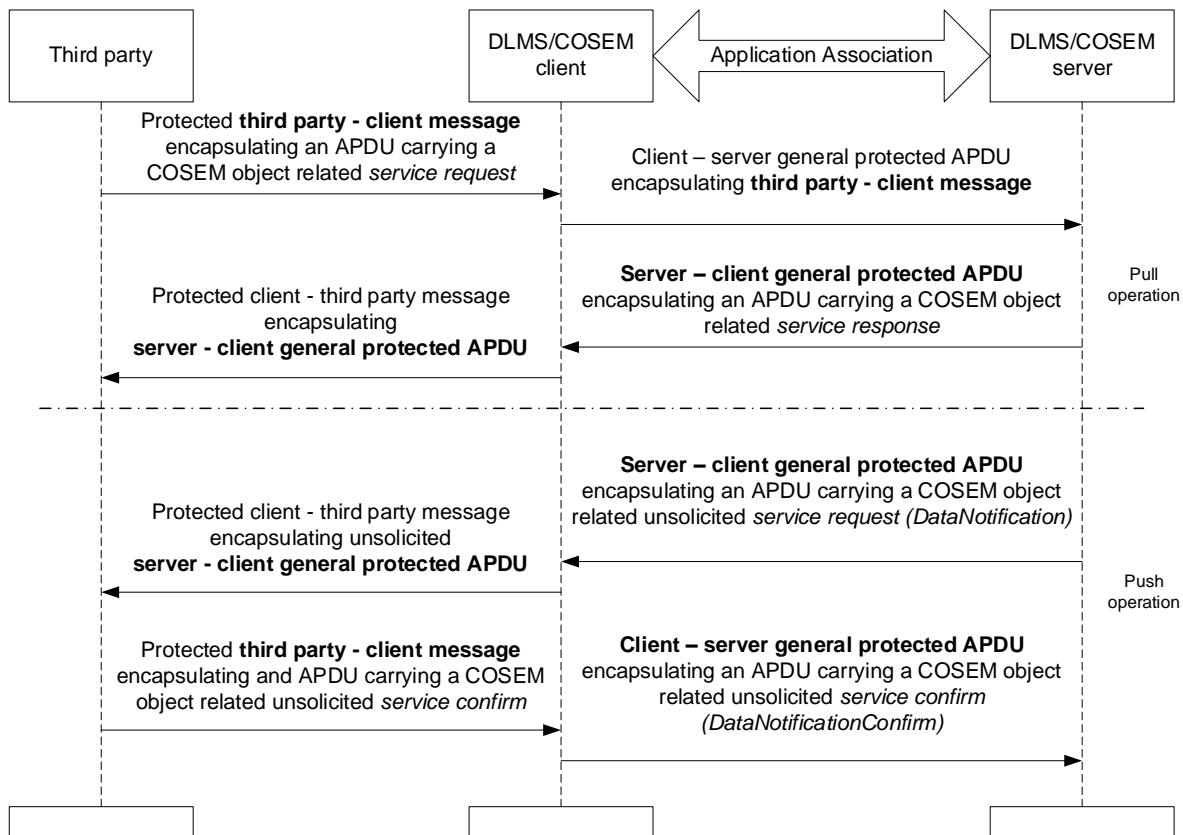
The concept of message protection between a client and a server is shown in Figure 90.



**Figure 90 – Client – server message security concept**

To ensure end-to-end message security, third parties have to be able to exchange protected xDLMS service requests with DLMS servers. In this case, the client acts as a broker, meaning that a third party is a user of one of the AAs between a client and a server the third party wants to reach.

The concept of message protection between a third party and a server is shown in Figure 91.

**Figure 91 – End-to-end message security concept**

The third party:

- is DLMS/COSEM aware i.e. it can generate and process messages encapsulating xDLMS APDUs carrying COSEM object related service requests and responses;
- it is able to apply its own protection to the xDLMS APDU carrying the request;
- it is able to verify protection applied by the server and / or the client on the response.

The DLMS client:

- acts as a broker between the third party and the server;
  - makes an appropriate AA available for use by the third party, based on information included in the TP – client message;
  - verifies that the TP has the right to use that AA;
- NOTE 2 The way to verify this is outside the Scope of this Technical Report.
- it may verify the protection applied by the third party;
  - encapsulates the third party – client message into a general protected xDLMS APDU;
  - it may verify the protection applied by the server on the APDU encapsulating the COSEM object related service response or unsolicited service request; (in the case of Push operation);
  - it may apply its own protection to the protected xDLMS APDUs sent to the TP.

The server:

- shall (pre-)establish an AA with the client used by the third party;
- it may check the identity of the third party using the AA;
- it shall provide access to COSEM object attributes and methods as determined by the security policy and access rights once the protection(s) applied by the client and/or the third party have been successfully verified;
- it shall prepare the response – or, in the case of Push operation an unsolicited service request – and apply the protection determined by the protection applied on the incoming request, the access rights and the security policy.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	209/614
-----------------------	------------	-----------------------	---------

The application of cryptographic protection on xDLMS APDUs is specified in 9.2.7.2.

### 9.2.2.6 COSEM data security

COSEM data i.e. values of COSEM object attributes, method invocation parameters and return parameters can be also cryptographically protected. When this is required, the attributes and methods concerned are accessed indirectly, via “Data protection” objects, that apply and verify / remove protection on COSEM data; see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.9.

See also 9.2.7.5.

### 9.2.3 Cryptographic algorithms

#### 9.2.3.1 Overview

DLMS/COSEM applies cryptography to protect the information.

NOTE The following text is quoted from NIST SP 800-21:2005, 3.1.

Cryptography is a branch of mathematics that is based on the transformation of data and can be used to provide several security services: confidentiality, data integrity, authentication, authorization and non-repudiation. Cryptography relies upon two basic components: an *algorithm* (or cryptographic methodology) and a *key*. The algorithm is a mathematical function, and the key is a parameter used in the transformation.

A cryptographic algorithm and key are used to apply cryptographic protection to data (e.g., encrypt the data or generate a digital signature) and to remove or check the protection (e.g., decrypt the encrypted data or verify the digital signature). There are three basic types of approved cryptographic algorithms:

- cryptographic hash functions that do not require keys (although they can be used in a mode in which keys are used). A hash function is often used as a component of an algorithm to provide a security service. See 9.2.3.2;
- symmetric key algorithms (often called secret key algorithms) that use a single key – shared by a sender and a receiver – to both apply the protection and to remove or check the protection. Symmetric key algorithms are relatively easy to implement and provide a high throughput. See 9.2.3.3;
- asymmetric key algorithms (often called public key algorithms) that use two keys (i.e., a key pair): a public key and a private key that are mathematically related to each other. Compared to symmetric key algorithms, implementation of asymmetric key algorithms is complex and requires much more computation. See 9.2.3.4.

In order to use cryptography, cryptographic keys must be “in place”, i.e., keys must be established for parties using cryptography. See 9.2.4.

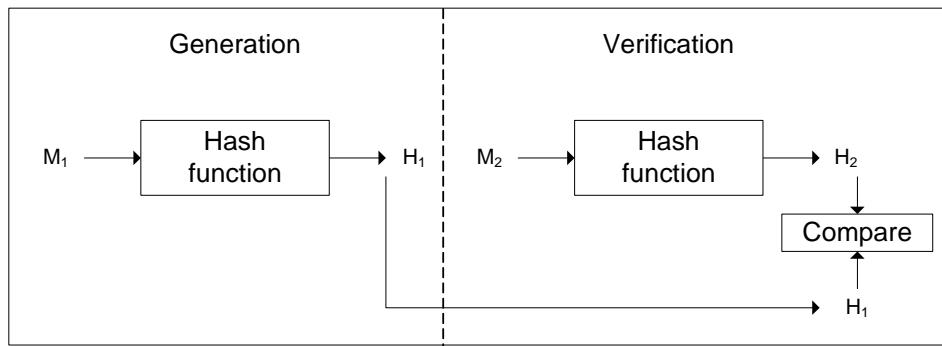
#### 9.2.3.2 Hash function

NOTE The following text is quoted from NIST SP 800-21:2005, 3.2.

A hash function produces a short representation of a longer message. A good hash function is a one-way function: it is easy to compute the hash value from a particular input; however, backing up the process from the hash value back to the input is extremely difficult. With a good hash function, it is also extremely difficult to find two specific inputs that produce the same hash value. Because of these characteristics, hash functions are often used to determine whether or not data has changed.

A hash function takes an input of arbitrary length and outputs a fixed length value. Common names for the output of a hash function include *hash value* and *message digest*. Figure 92 depicts the use of a hash function.

A hash value (H1) is computed on data (M1). M1 and H1 are then saved or transmitted. At a later time, the correctness of the retrieved or received data is checked by labelling the received data as M2 (rather than M1) and computing a new hash value (H2) on the received value. If the newly computed hash value (H2) is equal to the retrieved or received hash value (H1), then it can be assumed that the retrieved or received data (M2) is the same as the original data (M1) (i.e., M1 = M2).

**Figure 92 – Hash function**

Hash algorithms are used in DLMS/COSEM for the following purposes:

- digital signature, see 9.2.3.4.4;
- key agreement, see 9.2.3.4.6; and
- HLS authentication. The algorithm to be used depends on the authentication mechanism, see 9.2.7.4.

For digital signature and key agreement the algorithm shall be as stipulated by the security suite, see Table 27.

### 9.2.3.3 Symmetric key algorithms

#### 9.2.3.3.1 General

Symmetric key algorithms are used in DLMS/COSEM for the following purposes:

- authentication of communicating partners using HLS authentication mechanisms, see 9.2.7.4;
- authentication and encryption of xDLMS messages, see 9.2.7.2;
- authentication and encryption of COSEM data, see 9.2.7.5.

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.

Symmetric key algorithms (often called secret key algorithms) use a single key to both apply the protection and to remove or check the protection. For example, the key used to encrypt data is also used to decrypt the encrypted data. This key must be kept secret if the data is to retain its cryptographic protection. Symmetric key algorithms are used to provide confidentiality via encryption, or an assurance of authenticity or integrity via authentication, or are used during key establishment.

Keys used for one purpose shall not be used for other purposes. (See NIST SP 800-57:2012).

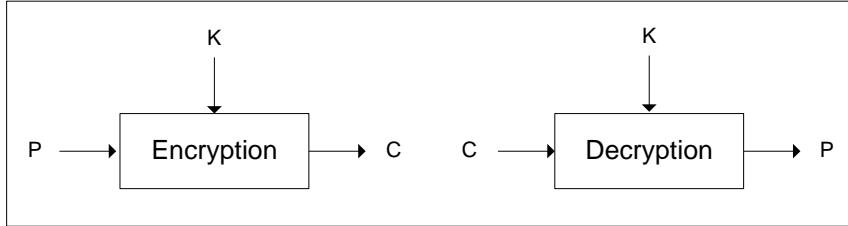
#### 9.2.3.3.2 Encryption and decryption

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.1.

Encryption is used to provide confidentiality for data. The data to be protected is called *plaintext*. Encryption transforms the data into *ciphertext*. Ciphertext can be transformed back into plaintext using decryption.

Plaintext data can be recovered from ciphertext only by using the same key that was used to encrypt the data. Unauthorized recipients of the ciphertext who know the cryptographic algorithm but do not have the correct key should not be able to decrypt the ciphertext. However, anyone who has the key and the cryptographic algorithm can easily decrypt the ciphertext and obtain the original plaintext data.

Figure 93 depicts the encryption and decryption processes. The plaintext ( $P$ ) and a key ( $K$ ) are used by the encryption process to produce the ciphertext ( $C$ ). To decrypt, the ciphertext ( $C$ ) and the same key ( $K$ ) are used by the decryption process to recover the plaintext ( $P$ ).

**Figure 93 – Encryption and decryption**

With symmetric key block cipher algorithms, the same plaintext block and key will always produce the same ciphertext block. This property does not provide acceptable security. Therefore, cryptographic modes of operation have been defined to address this problem (see 9.2.3.3.4).

#### **9.2.3.3.3 Advanced Encryption Standard**

For the purposes of DLMS/COSEM, the Advanced Encryption Standard (AES) as specified in FIPS PUB 197:2001 shall be used. AES operates on blocks (chunks) of data during an encryption or decryption operation. For this reason, AES is referred to as a block cipher algorithm.

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.1.3.

AES encrypts and decrypts data in 128-bit blocks, using 128, 192 or 256 bit keys. All three key sizes are adequate.

AES offers a combination of security, performance, efficiency, ease of implementation, and flexibility. Specifically, the algorithm performs well in both hardware and software across a wide range of computing environments. Also, the very low memory requirements of the algorithm make it very well suited for restricted-space environments.

#### **9.2.3.3.4 Encryption Modes of Operation**

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.1.4.

With a symmetric key block cipher algorithm, the same plaintext block will always encrypt to the same ciphertext block when the same symmetric key is used. If the multiple blocks in a typical message (data stream) are encrypted separately, an adversary could easily substitute individual blocks, possibly without detection. Furthermore, certain kinds of data patterns in the plaintext, such as repeated blocks, would be apparent in the ciphertext.

Cryptographic modes of operation have been defined to address this problem by combining the basic cryptographic algorithm with variable initialization values (commonly known as initialization vectors) and feedback rules for the information derived from the cryptographic operation.

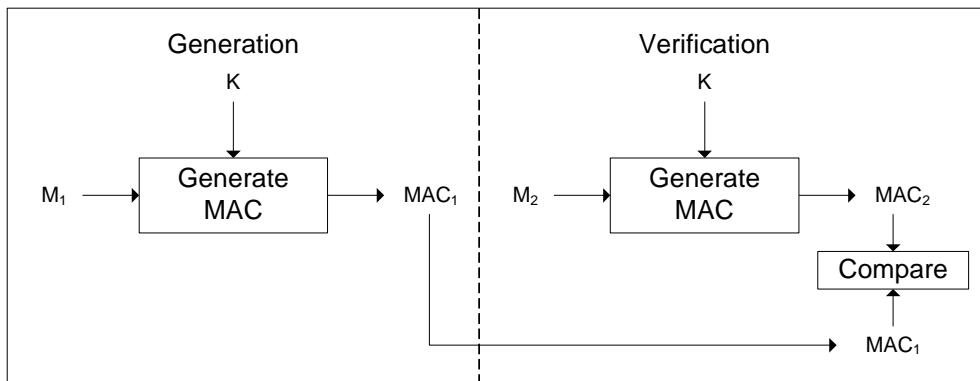
NIST SP 800-38D:2007 specifies the Galois/Counter Mode (GCM), an algorithm for authenticated encryption with associated data, and its specialization, GMAC, for generating a message authentication code (MAC) on data that is not encrypted. GCM and GMAC are modes of operation for an underlying approved symmetric key block cipher. See 9.2.3.3.3.

#### **9.2.3.3.5 Message Authentication Code**

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.2.

Message Authentication Codes (MACs) provide an assurance of authenticity and integrity. A MAC is a cryptographic checksum on the data that is used to provide assurance that the data has not changed or been altered and that the MAC was computed by the expected party (the sender). Typically, MACs are used between two parties that share a secret key to authenticate information exchanged between those parties.

Figure 94 depicts the use of message authentication codes (MACs).



**Figure 94 – Message Authentication Codes (MACs)**

A MAC (MAC1) is computed on data (M1) using a key (K). M1 and MAC1 are then saved or transmitted. At a later time, the authenticity of the retrieved or received data is checked by labelling the retrieved or received data as M2 and computing a MAC (MAC2) on it using the same key (K). If the retrieved or received MAC (MAC1) is the same as the newly computed MAC (MAC2), then it can be assumed that the retrieved or received data (M2) is the same as the original data (M1) (i.e., M1 = M2). The verifying party also knows who the sending party is because no one else knows the key.

Typically, MACs are used to detect data modifications that occur between the initial generation of the MAC and the verification of the received MAC. They do not detect errors that occur before the MAC is originally generated.

Message integrity is frequently provided using non-cryptographic techniques known as error detection codes. However, these codes can be altered by an adversary to the adversary's benefit. The use of an approved cryptographic mechanism, such as a MAC, addresses this problem. That is, the integrity provided by a MAC is based on the assumption that it is not possible to generate a MAC without knowing the cryptographic key. An adversary without knowledge of the key will be unable to modify data and then generate an authentic MAC on the modified data. It is therefore crucial that MAC keys be kept secret.

For the purposes of DLMS/COSEM, the GMAC algorithm as specified in 9.2.3.3.7.2 shall be used.

#### 9.2.3.3.6 Key wrapping

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.3.

Symmetric key algorithms may be used to wrap (i.e., encrypt) keying material using a key-wrapping key (also known as a key encrypting key). The wrapped keying material can then be stored or transmitted securely. Unwrapping the keying material requires the use of the same key-wrapping key that was used during the original wrapping process.

Key wrapping differs from simple encryption in that the wrapping process includes an integrity feature. During the unwrapping process, this integrity feature detects accidental or intentional modifications to the wrapped keying material. For the purposes of DLMS/COSEM the AES key wrap algorithm shall be used; see 9.2.3.3.8.

#### 9.2.3.3.7 Galois/Counter Mode

##### 9.2.3.3.7.1 General

NOTE The following text is taken from NIST SP 800-38D:2007, Clause 3.

Galois/Counter Mode (GCM) is an algorithm for authenticated encryption with associated data. GCM is constructed from an approved symmetric key block cipher with a block size of 128 bits, such as the Advanced Encryption Standard (AES) algorithm, see FIPS PUB 197. Thus, GCM is a mode of operation of the AES algorithm.

GCM provides assurance of the confidentiality of data using a variation of the Counter mode of operation for encryption.

GCM provides assurance of the authenticity of the confidential data (up to about 64 gigabytes per invocation) using a universal hash function that is defined over a binary Galois (i.e., finite) field (GHASH). GCM can also provide authentication assurance for additional data (of practically unlimited length per invocation) that is not encrypted.

If the GCM input is restricted to data that is not to be encrypted, the resulting specialization of GCM, called GMAC, is simply an authentication mode on the input data.

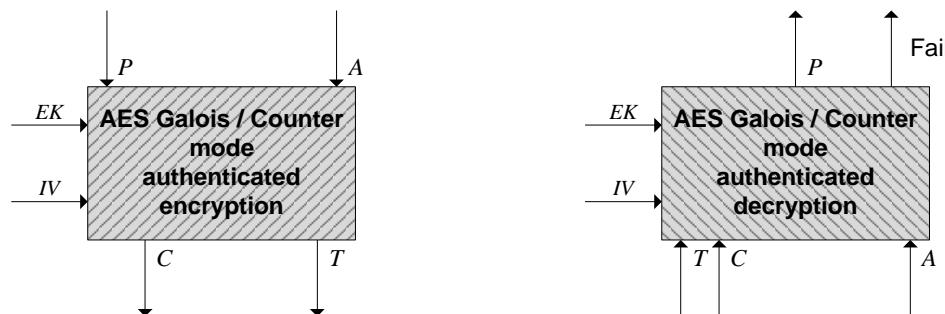
GCM provides stronger authentication assurance than a (non-cryptographic) checksum or error detecting code; in particular, GCM can detect both 1) accidental modifications of the data and 2) intentional, unauthorized modifications.

In DLMS/COSEM, it is also possible to use GCM to provide confidentiality only: in this case, the authentication tags are simply not computed and checked.

#### 9.2.3.3.7.2 GCM functions

NOTE The following is based on NIST SP 800-38D:2007, 5.2.

The two functions that comprise GCM are called authenticated encryption and authenticated decryption; see Figure 95.



**Figure 95 – GCM functions**

The authenticated encryption function encrypts the confidential data and computes an authentication tag on both the confidential data and any additional, non-confidential data. The authenticated decryption function decrypts the confidential data, contingent on the verification of the tag.

When the input is restricted to non-confidential data, the resulting variant of GCM is called GMAC. For GMAC, the authenticated encryption and decryption functions become the functions for generating and verifying an authentication tag on the non-confidential data.

Finally, if authentication is not required, the authenticated encryption function encrypts the confidential data, but the authentication tag is not computed. The authenticated decryption function decrypts the confidential data, but no authentication tag is computed and verified.

In DLMS/COSEM, the use of authentication and encryption is indicated by bit 4 and bit 5 of the Security Control Byte, specified in 9.2.7.2.4.

a) The authenticated encryption function – given the selection of a block cipher key  $EK$  – has three input strings:

- a plaintext, denoted  $P$ ;
- Additional Authenticated Data (AAD), denoted  $A$ ;
- an initialization vector (IV) denoted  $IV$ .

The plaintext and the AAD are the two categories of data that GCM protects. GCM protects the authenticity of the plaintext and the AAD; GCM also protects the confidentiality of the plaintext, while the AAD is left in the clear.

The IV is essentially a nonce, i.e. a value that is unique within the specified (security) context, which determines an invocation of the authenticated encryption function on the input data to be protected. See 9.2.3.3.7.3.

The bit lengths of the input strings to the authenticated encryption function shall meet the following requirements:

- $\text{len}(P) < 2^{39}-256$ ;
- $\text{len}(A) < 2^{64}-1$ ;
- $1 \leq \text{len}(IV) \leq 2^{64}-1$ .

The bit lengths of  $P$ ,  $A$  and  $IV$  shall all be multiples of 8, so that these values are byte strings.

There are two outputs:

- a ciphertext, denoted  $C$  whose bit length is the same as that of the plaintext  $P$ ;
- an authentication tag, or tag, for short, denoted  $T$ .

b) The authenticated decryption function – given the selection of a block cipher key  $EK$  – has four input strings:

- the initialization vector, denoted  $IV$ ;
- the ciphertext, denoted  $C$ ;
- the Additional Authenticated Data (AAD), denoted  $A$ ;
- the authentication tag, denoted  $T$ .

The output is one of the following:

- the plaintext  $P$  that corresponds to the ciphertext  $C$ , or
- a special error code denoted *FAIL* in this Technical Report.

The output  $P$  indicates that  $T$  is the correct authentication tag for  $IV$ ,  $A$ , and  $C$ ; otherwise, the output is *FAIL*.

### 9.2.3.3.7.3 The initialization vector, $IV$

In DLMS/COSEM, for the construction of the initialization vector  $IV$  deterministic construction as specified in NIST SP 800-38D:2007, 8.2.1 shall be used: the  $IV$  is the concatenation of two fields, called the fixed field and the invocation field. The fixed field shall identify the physical device, or, more generally, the (security) context for the instance of the authenticated encryption function. The invocation field shall identify the sets of inputs to the authenticated encryption function in that particular device.

For any given key, no two distinct physical devices shall share the same fixed field, and no two distinct sets of inputs to any single device shall share the same invocation field.

The length of the  $IV$  shall be 96 bits (12 octets):  $\text{len}(IV) = 96$ . Within this:

- the leading (i.e. the leftmost) 64 bits (8 octets) shall hold the fixed field. It shall contain the system title, see 4.3.4;
- the trailing (i.e. the rightmost) 32 bits shall hold the invocation field. The invocation field shall be an integer counter.

Each encryption key ( $EK$ ) has two invocation counters ( $IC$ ) associated with it, one for the authenticated encryption function and the other for the authenticated decryption function. The  $EK$  is for block cyphering. The following rules apply:

- when the key is established, the corresponding IC's are reset to 0;
- when the authenticated encryption function is used, the corresponding IC is used after which it is incremented by 1. If the maximum value of the IC has been reached, any further invocations of the authenticated encryption function shall return an error and the IC shall not be incremented.
- When the authenticated decryption function is used, the value of the  $IC$  is verified. The value must be equal to or greater than the lowest acceptable value.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	215/614
-----------------------	------------	-----------------------	---------

If the value being verified satisfies this requirement, the lowest acceptable value is set to the  $/C$  value verified plus 1 following the use of the authentication decryption function.

If the value being verified is less than the lowest acceptable value, the verification fails and so does the authenticated decryption function.

If the value being verified is equal to the maximum value, the authenticated decryption function shall return an error.

NOTE The maximum number of invocations is  $2^{32}-1$ .

The bit length of the fixed field limits the number of distinct physical devices that can implement the authenticated encryption function for the given key to  $2^{64}$ . The bit length of the invocation field limits the number of invocations of the authenticated encryption function to  $2^{32}$  with any given input sets without violating the uniqueness requirement.

#### 9.2.3.3.7.4 The encryption key, $EK$

GCM uses a single key, the block cipher key. In DLMS/COSEM, this is known as the encryption key, denoted  $EK$ . Its size depends on the security suite – see 9.2.3.7 – and shall be:

- for security suite 0 and 1, 128 bits (16 octets):  $\text{len}(EK) = 128$ ;
- for security suite 2, 256 bits (32 octets):  $\text{len}(EK) = 256$ ;

The key shall be generated uniformly at random, or close to uniformly at random, i.e., so that each possible key is (nearly) equally likely to be generated. Consequently, the key will be fresh, i.e., unequal to any previous key, with high probability. The key shall be secret and shall be used exclusively for GCM with the chosen block cipher AES. Additional requirements on the establishment and management of keys are discussed in NIST SP 800-38D:2007, 8.1.

#### 9.2.3.3.7.5 The authentication key, $AK$

In DLMS/COSEM, for additional security, an authentication key denoted  $AK$  is also specified. When present, it shall be part of the Additional Authenticated Data, AAD. For its length and its generation, the same rules apply as for the encryption key.

#### 9.2.3.3.7.6 Length of the authentication tag

The bit length of the authentication tag, denoted  $t$ , is a security parameter. In security suites 0, 1 and 2 its value shall be 96 bits.

#### 9.2.3.3.8 AES key wrap

For wrapping key data DLMS/COSEM has selected the AES key wrap algorithm specified in RFC 3394. The algorithm is designed to wrap or encrypt key data. It operates on blocks of 64 bits. Before being wrapped, the key data is parsed into  $n$  blocks of 64 bits. The only restriction the key wrap algorithm places on  $n$  is that  $n$  has to be at least two.

The AES key wrap can be configured to use any of the three key sizes supported by the AES codebook: 128, 192, 256.

The two algorithms are key wrap and key unwrap.

The inputs to the key wrapping process are the Key Encrypting Key  $KEK$  and the plaintext to be wrapped. The plaintext consists of  $n$  64-bit blocks, containing the key data being wrapped. The output is the ciphertext,  $(n+1)$  64 bit values.

The inputs to the unwrap process are the  $KEK$  and  $(n+1)$  64-bit blocks of ciphertext consisting of previously wrapped key. It returns  $n$  blocks of plaintext consisting of the  $n$  64-bit blocks of the decrypted key data.

In DLMS/COSEM, the size of  $KEK$  depends on the security suite – see 9.2.3.7 – and shall be:

- for security suite 0 and 1, 128 bits (16 octets):  $\text{len}(KEK) = 128$ ;
- for security suite 2, 256 bits (32 octets):  $\text{len}(KEK) = 256$ .

### 9.2.3.4 Public key algorithms

#### 9.2.3.4.1 General

In general, public key cryptography systems use hard-to-solve problems as the basis of the algorithm. The RSA algorithm is based on the prime factorization of very large integers. Elliptic Curve Cryptography (ECC) is based on the difficulty of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP). ECC provides similar levels of security compared to RSA but with significantly reduced key sizes. ECC is particularly suitable for embedded devices and therefore it has been selected for use in DLMS/COSEM.

Public key algorithms are used in DLMS/COSEM for the following purposes:

- authentication of communicating partners;
- digital signature of xDLMS APDUs and COSEM data;
- key agreement.

NOTE 1 The following text is quoted from NIST SP 800-21:2005, 3.4.

Asymmetric key algorithms (often called public key algorithms) use two keys: a public key and a private key, which are mathematically related to each other. The public key may be made public; the private key must remain secret if the data is to retain its cryptographic protection. Even though there is a relationship between the two keys, the private key cannot be determined from the public key. Which key to be used to apply versus remove or check the protection depends on the service to be provided. For example, a digital signature is computed using a private key and the signature is verified using the public key; for those algorithms also capable of encryption, the encryption is performed using the public key, and the decryption is performed using the private key.

NOTE 2 Not all public key algorithms are capable of multiple functions, e.g., generating digital signatures and encryption. Asymmetric key algorithms are not used for encryption in DLMS/COSEM.

Asymmetric key algorithms are used primarily as data integrity, authentication, and non-repudiation mechanisms (i.e., digital signatures), as well as for key establishment.

Some asymmetric key algorithms use domain parameters, which are additional values necessary for the operation of the cryptographic algorithm. These values are mathematically related to each other. Domain parameters are usually public and are used by a community of users for a substantial period of time.

The secure use of asymmetric key algorithms requires that users obtain certain assurances:

- assurance of domain parameter validity provides confidence that the domain parameters are mathematically correct;
- assurance of public key validity provides confidence that the public key appears to be a suitable key; and
- assurance of private key possession provides confidence that the party that is supposedly the owner of the private key really has the key.

Some asymmetric key algorithms may be used for multiple purposes (e.g., for both digital signatures and key establishment). Keys used for one purpose shall not be used for other purposes.

#### 9.2.3.4.2 Elliptic curve cryptography

##### 9.2.3.4.2.1 General

Elliptic curve cryptography involves arithmetic operations on an elliptic curve over a finite field. Elliptic curves can be defined over any field of numbers (i.e., real, integer, complex) although they are most often used over finite prime fields for applications in cryptography.

An elliptic curve on a prime field consists of the set of real numbers ( $x, y$ ) that satisfy the equation:

$$y^2 = x^3 + ax + b$$

The set of all of the solutions to the equation forms the elliptic curve. Changing  $a$  and  $b$  changes the shape of the curve, and small changes in these parameters can result in major changes in the set of ( $x, y$ ) solutions.

#### 9.2.3.4.2.2 NIST recommended elliptic curves

FIPS PUB 186-4:2013 recommends five prime field elliptic curves over a prime field  $GF(p)$ . Of these, the curves P-256 and P-384 have been selected for DLMS/COSEM as shown in Table 21.

**Table 21 – Elliptic curves in DLMS/COSEM security suites**

DLMS security suite	Curve name in FIPS PUB 186-4:2013	ASN.1 Object Identifier
Suite 0	–	–
Suite 1	NIST curve P-256	1.2.840.10045.3.1.7
Suite 2	NIST curve P-384	1.3.132.0.34

NOTE The ASN.1 Object Identifier appears in the Certificate under AlgorithmIdentifier: Parameters. See 9.2.6.4.2.

#### 9.2.3.4.3 Data conversions

##### 9.2.3.4.3.1 Overview

This clause describes the data conversion primitives that shall be used to convert between different data types used to specify public key algorithms: octet strings (OS), bit strings (BS), integers (I), field elements (FE) and elliptic curve points (ECP). DLMS/COSEM uses octet strings to represent elements of public key algorithms and uses conversion primitives between these data types from and to octet strings. The octet string  $M_{d-1} M_{d-2} \dots M_0$  of length  $d$  is encoded as A-XDR OCTET STRING where the leftmost octet  $M_{d-1}$  corresponds to first octet of the encoded value of the OCTET STRING.

##### 9.2.3.4.3.2 Conversion between Bit Strings and Octet Strings (BS2OS)

The data conversion primitive that converts a bit string to an octet string is called the Bit String to Octet String Conversion Primitive, or BS2OS. It takes the bit string as input and outputs the octet string. The bit string  $b_{l-1} b_{l-2} \dots b_0$  of length  $l$  shall be converted to an octet string  $M_{d-1} M_{d-2} \dots M_0$  of length  $d = \lceil l/8 \rceil$ .

The conversion pads enough zeroes on the left to make the number of bits multiple of eight, and then breaks it into octets.

More precisely, conversion shall be as follows:

- for  $0 \leq i < d - 1$ , let the octet  $M_i = b_{8i+7} b_{8i+6} \dots b_{8i}$ ;
- the leftmost octet  $M_{d-1}$  shall have its leftmost  $8d - l$  bits set to zero;
- its rightmost  $8 - (8d - l)$  bits shall be  $b_{l-1} b_{l-2} \dots b_{8d-8}$ .

##### 9.2.3.4.3.3 Conversion between Octet Strings and Bit Strings (OS2BS)

The data conversion primitive that converts an octet string to a bit string is called the Octet String to Bit String Conversion Primitive, or OS2BS. It takes the octet string as input and outputs the bit string. The octet string  $M_{d-1} M_{d-2} \dots M_0$  of length  $d$  shall be converted to a bit string  $b_{l-1} b_{l-2} \dots b_0$  of desired length  $l$ , where  $d = \lceil l/8 \rceil$  and the leftmost  $8d-l$  bits of the leftmost octet are zero.

More precisely conversion shall be as follows:

- for  $0 \leq i < l - 1$ , let the bits  $b_{8i+7} b_{8i+6} \dots b_{8i} = M_i$ ;
- its leftmost  $(8d - l)$  bits of the leftmost octet shall be zero.

##### 9.2.3.4.3.4 Conversion between Integers and Octet Strings (I2OS)

The data conversion primitive that converts an integer to an octet string is called the Integer to Octet String Conversion Primitive, or I2OS. It takes a non-negative integer  $x$  and the desired length  $d$  of the octet string as input. The length  $d$  has to satisfy  $256^d > x$ , otherwise it shall output “error”. I2OS outputs the corresponding octet string.

The integer  $x$  shall be written in its unique  $l$ -digit representation base 256:

218/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

- $x = x_{d-1} \cdot 256^{d-1} + x_{d-2} \cdot 256^{d-2} + \dots + x_1 \cdot 256 + x_0;$
- where  $0 \leq x_i < 256$  for  $0 \leq i \leq d-1$ ;
- $M_i = x_i$ , for  $0 \leq i \leq d-1$ .

The output octet string shall be  $M_{d-1} M_{d-2} \dots M_0$ .

#### 9.2.3.4.3.5 Conversion between Octet Strings and Integers (OS2I)

The data conversion primitive that converts an octet string to an integer is called the Octet String to Integer Conversion Primitive, or OS2I. It takes the octet string  $M_{d-1} M_{d-2} \dots M_0$  of length  $d$  as an input and outputs the corresponding integer  $x$ . In the case of the octet string of length zero, the conversion outputs integer 0.

Each octet is interpreted as a non-negative integer to the base 256. More precisely, conversion shall be as follows:

- $x_i = M_i$ , for  $0 \leq i \leq d-1$ ;
- $x = x_{d-1} \cdot 256^{d-1} + x_{d-2} \cdot 256^{d-2} + \dots + x_1 \cdot 256 + x_0$ .

#### 9.2.3.4.3.6 Conversion between Field Elements and Octet Strings (FE2OS)

The data conversion primitive that converts a field element to an octet string is called the Field Element to Octet String Conversion Primitive, or FE2OS. It takes a field element as input and outputs the corresponding octet string. A field element  $x \in F_p$  is converted to an octet string  $M_{d-1} M_{d-2} \dots M_0$  of length  $d = \lceil \log_{256} p \rceil$  by applying I2OS conversion primitive with parameter  $l$ , where

- $\text{FE2OS}(x) = \text{I2OS}(x, l)$ .

#### 9.2.3.4.3.7 Conversion between Octet Strings and Field Elements (OS2FE)

The data conversion primitive that converts an octet string to a field element is called the Octet String to Field Element Conversion Primitive, or OS2FE. It takes an octet string as input and outputs the corresponding field element. An octet string  $M_{d-1} M_{d-2} \dots M_0$  of length  $d$  is converted to field element  $x \in F_p$  by applying OS2I conversion primitive where:

- $\text{OS2FE}(x) = \text{OS2I}(x) \bmod p$ .

#### 9.2.3.4.4 Digital signature

NOTE The following text is quoted from NIST SP 800-21:2005, 3.4.1.

A digital signature is an electronic analogue of a written signature that can be used in proving to the recipient or a third party that the message was signed by the originator (a property known as non-repudiation). Digital signatures may also be generated for stored data and programs so that the integrity of the data and programs may be verified at a later time.

Digital signatures authenticate the integrity of the signed data and the identity of the signatory. A digital signature is represented in a computer as a string of bits and is computed using a digital signature algorithm that provides the capability to generate and verify signatures. Signature generation uses a private key to generate a digital signature. Signature verification uses the public key that corresponds to, but is not the same as, the private key to verify the signature. Each signatory possesses a private and public key pair. Signature generation can be performed only by the possessor of the signatory's private key. However, anyone can verify the signature by employing the signatory's public key.

The security of a digital signature system is dependent on maintaining the secrecy of a signatory's private key. Therefore, users must guard against the unauthorized acquisition of their private keys.

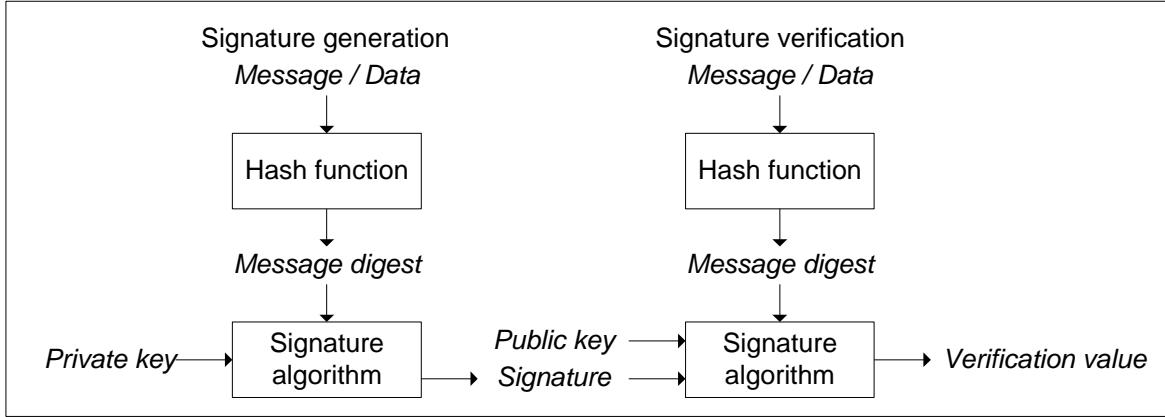
**Figure 96 – Digital signatures**

Figure 96 depicts the digital signature process. A hash function (see 9.2.3.2) is used in the signature generation process to obtain a condensed version of data to be signed, called a message digest or hash value. The message digest is then input to the digital signature algorithm to generate the digital signature. The digital signature is sent to the intended verifier along with the signed data (often called the message). The verifier of the message and signature verifies the signature by using the signatory's public key. The same hash function and digital signature algorithm must also be used in the verification process. Similar procedures may be used to generate and verify signatures for stored as well as transmitted data.

#### 9.2.3.4.5 Elliptic curve digital signature (ECDSA)

For DLMS/COSEM the elliptic curve digital signature (ECDSA) algorithm as specified in FIPS PUB 186-4:2013 has been selected. NSA1 provides an implementation guide.

In DLMS/COSEM the elliptic curves and algorithms used shall be:

- in the case of Security Suite 1, the elliptic curve P-256 with the SHA-256 hash algorithm;
- in the case of Security Suite 2, the elliptic curve P-384 with the SHA-384 hash algorithm.

The inputs to ECDSA digital signature generation are the following:

- the message  $M$  to be signed;  
NOTE 1 In the DLMS/COSEM context "Message" may be an xDLMS APDU, see 9.2.7.2 or COSEM data, see 9.2.7.5.
- the private key of the signatory,  $d$ .

The output is the ECDSA signature  $(r, s)$  over  $M$ .

In DLMS/COSEM the plain format shall be used: the signature  $(r, s)$  is encoded as octet string  $R \parallel S$ , i.e. as concatenation of the octet strings  $R = \text{I2OS}(r, l)$  and  $S = \text{I2OS}(s, l)$  with  $l = \lceil \log_{256} n \rceil$ . Thus, the signature has a fixed length of  $2l$  octets.

NOTE 2 Here,  $n$  is the order of the base point  $G$  of the elliptic curve. I2OS is the Integer to Octet String Conversion Primitive. See 9.2.3.4.3.

The inputs to the verification of the ECDSA digital signature generation are the following:

- the signed message  $M'$ ;
  - the received ECDSA signature  $(r', s')$ ;
  - the authentic public key of the signatory,  $Q$ .
- The process of generating and verifying the signatures shall be as specified in NSA1, 3.4.

#### 9.2.3.4.6 Key agreement

##### 9.2.3.4.6.1 Overview

Key agreement allows two entities to jointly compute a shared secret and derive secret keying material from it. See also NIST SP 800-56A Rev. 2: 2013.

For DLMS/COSEM three elliptic curve key agreement schemes have been selected from NIST SP 800-56A Rev. 2: 2013:

- the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme;
- the One-Pass Diffie-Hellman C(1e,1s, ECC CDH) scheme;
- the Static Unified Model C(0e, 2s, ECC CDH) scheme.

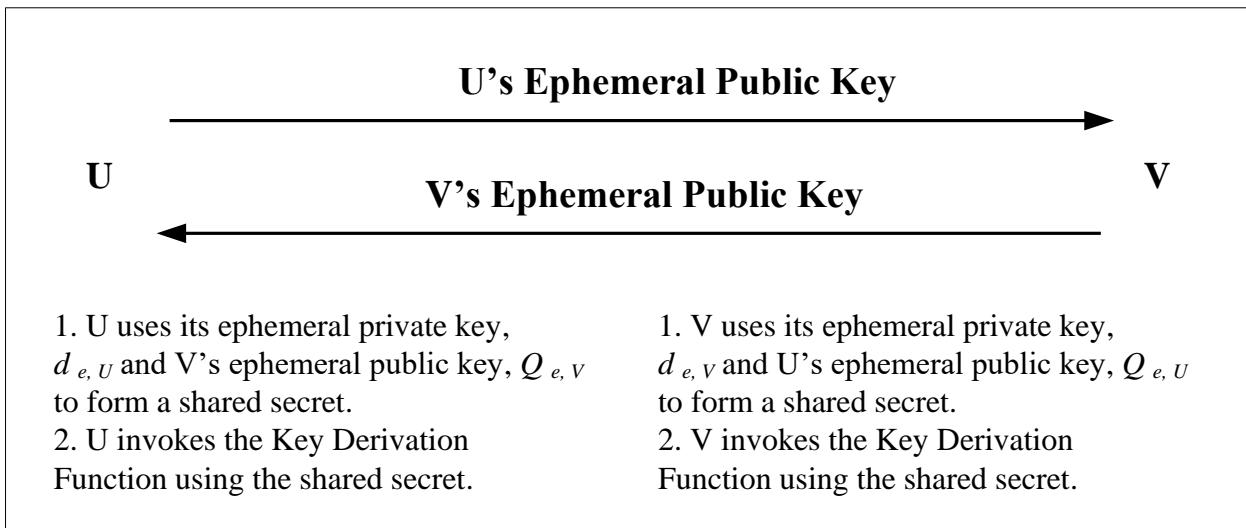
NOTE For NSA Suite B the first two schemes have been selected.

#### 9.2.3.4.6.2 The Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme

This scheme is for use between a DLMS client and a server to agree on the master key, on global encryption keys and/or on the authentication key. The client plays the role of party U and the server plays the role of party V. The process is supported by the methods of the “Security setup” interface class; see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7.

The parties generate an ephemeral key pair from the domain parameters  $D$ . The parties exchange ephemeral public keys and then compute the shared secret  $Z$  using the domain parameters, their ephemeral private key and the ephemeral public key of the other party. The secret keying material is derived using the key derivation function specified in 9.2.3.4.6.5 from the shared secret  $Z$  and other input.

The process is specified in detail in NIST SP 800-56A Rev. 2: 2013, 6.1.2.2 and NSA2, 3.1 and it is shown in Figure 97 and Table 22 below.



NOTE This figure reproduces NSA2, Figure 1.

**Figure 97 – C(2e, 0s) scheme: each party contributes only an ephemeral key pair**

Prerequisites:

- a) each party has an authentic copy of the same set of domain parameters,  $D$ .  $D$  shall be selected from one of the two sets of domain parameters, see Annex A;
- b) the parties have agreed on using the NIST Concatenation KDF; see 9.2.3.4.6.5. SHA-256 is the hash function to use with the domain parameters for P-256 and SHA-384 is the hash function to use with the domain parameters for P-384;
- c) prior to or during the key agreement process, the parties obtain the identifier associated with the other party during the key agreement scheme.

NOTE See also NIST SP 800-56A Rev. 2: 2013 and NSA2 for additional information on assurance on the validity of the domain parameters, the validity of the private and public key and the assurance of the possession of the private key.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	221/614
-----------------------	------------	-----------------------	---------

**Table 22 – Ephemeral Unified Model key agreement scheme summary**

	<b>Party U</b>	<b>Party V</b>
Domain parameters	$(q, FR, a, b\{SEED\}, G, n, h)$ as determined by the security suite	$(q, FR, a, b\{SEED\}, G, n, h)$ as determined by the security suite
Static data	N/A	N/A
Ephemeral data	Ephemeral private key, $d_{e, U}$ Ephemeral public key, $Q_{e, U}$	Ephemeral private key, $d_{e, V}$ Ephemeral public key, $Q_{e, V}$
Computation	Compute $Z$ by calling ECC CDH using $d_{e, U}$ and $Q_{e, V}$	Compute $Z$ by calling ECC CDH using $d_{e, V}$ and $Q_{e, U}$
Derive secret keying material	1. Compute $kdf(Z, OtherInput)$ 2. Destroy $Z$	1. Compute $kdf(Z, OtherInput)$ 2. Destroy $Z$

NOTE This table is based on NIST SP 800-56A Rev. 2: 2013, Table 18 and NSA2, Table 1.

The rationale for choosing a C(2e, 0s) scheme is specified in NIST SP 800-56A Rev. 2: 2013, 8.2.

The use of this scheme in DLMS/COSEM is further explained in C.1.

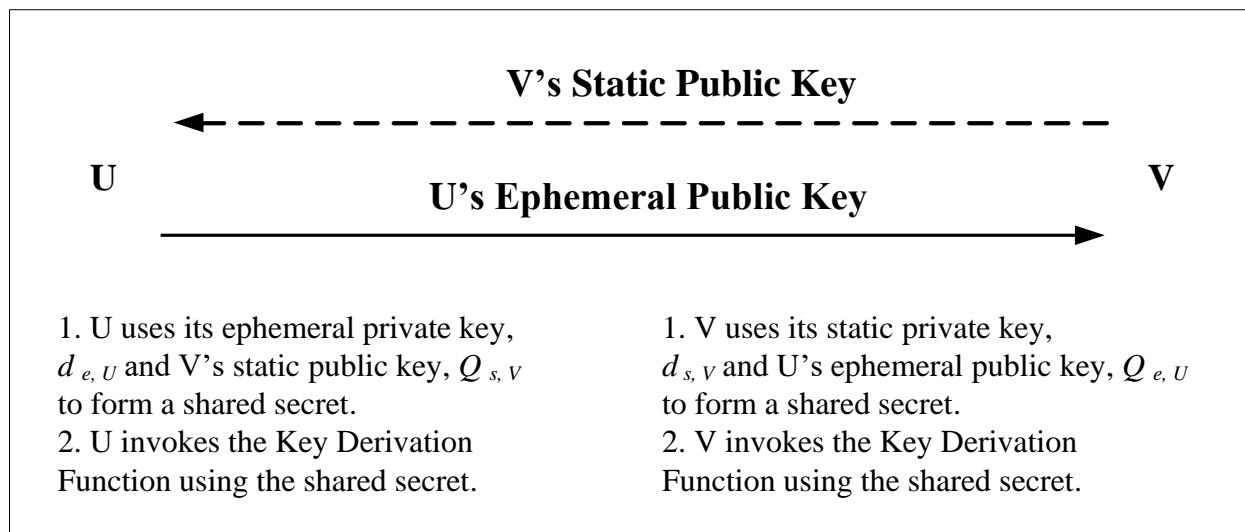
#### 9.2.3.4.6.3 The One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme

This scheme is for use by a DLMS server and another party to agree on an ephemeral encryption key to protect xDLMS APDUs or COSEM data. The party sending the message (the originator) plays the role of party U and the other party (the recipient) plays the role of party V.

NOTE 1 The terms originator and recipient refer to fields of the general-ciphering APDU.

For this scheme, party U generates an ephemeral key pair; party V has only a static key pair. Party U obtains Party V's static public key in a trusted manner (for example, from a certificate signed by a trusted CA) and sends its ephemeral public key to party V. Each party derives the shared secret Z by using its own private key and the other party's public key. Each party derives secret keying material using the key derivation method specified in 9.2.3.4.6.5 from the shared secret Z and other input.

The process is specified in detail in NIST SP 800-56A Rev. 2: 2013, 6.2.2.2 and NSA2, 3.2 and is shown in Figure 98 and Table 23.



NOTE This figure reproduces NSA2, Figure 2.

**Figure 98 – C(1e, 1s) schemes: party U contributes an ephemeral key pair, and party V contributes a static key pair**

Prerequisites:

222/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

- a) each party shall have an authentic copy of the same set of domain parameters, D. D shall be selected from one of the two sets of domain parameters, see Annex A;
- b) party V shall have been designated as the owner of a static key pair that was generated as specified in 9.2.6.2 using the set of domain parameters, D;
- c) the parties have agreed on using the NIST Concatenation KDF, see 9.2.3.4.6.5. SHA-256 is the hash function to use with the domain parameters for P-256 and SHA-384 is the hash function to use with the domain parameters for P-384;
- d) prior to or during the key agreement process, the parties obtain the identifier associated with the other party during the key agreement scheme. Party U shall obtain the static public key that is bound to party V's identifier. This static public key shall be obtained in a trusted manner (e.g., from a certificate signed by a trusted CA).

NOTE 2 See also NIST SP 800-56A Rev. 2: 2013 and NSA2 for additional information on assurance on the validity of the domain parameters, the validity of the private and public key and the assurance of the possession of the private key.

**Table 23 – One-pass Diffie-Hellman key agreement scheme summary**

	<b>Party U</b>	<b>Party V</b>
Domain Parameters	( $q, FR, a, b\{SEED\}, G, n, h$ ) As determined by the security suite	( $q, FR, a, b\{SEED\}, G, n, h$ ) As determined by the security suite
Static Data	N/A	Static private key, $d_{s, v}$ Static public key, $Q_{s, v}$
Ephemeral Data	Ephemeral private key, $d_{e, u}$ Ephemeral public key, $Q_{e, u}$	N/A
Computation	Compute Z by calling ECC CDH using $d_{e, u}$ and $Q_{s, v}$	Compute Z by calling ECC CDH using $d_{s, v}$ and $Q_{e, u}$
Derive Secret Keying Material	1. Compute kdf(Z, OtherInput) 2. Destroy Z	1. Compute kdf(Z, OtherInput) 2. Destroy Z

NOTE This table is based on NIST SP 800-56A Rev. 2: 2013, Table 24 and NSA2, Table 2.

The rationale for choosing this scheme is specified in NIST SP 800-56A Rev. 2: 2013, 8.4.

The use of this scheme in DLMS/COSEM is further explained in C.2.

#### 9.2.3.4.6.4 The Static Unified Model C(0e, 2s, ECC CDH) scheme

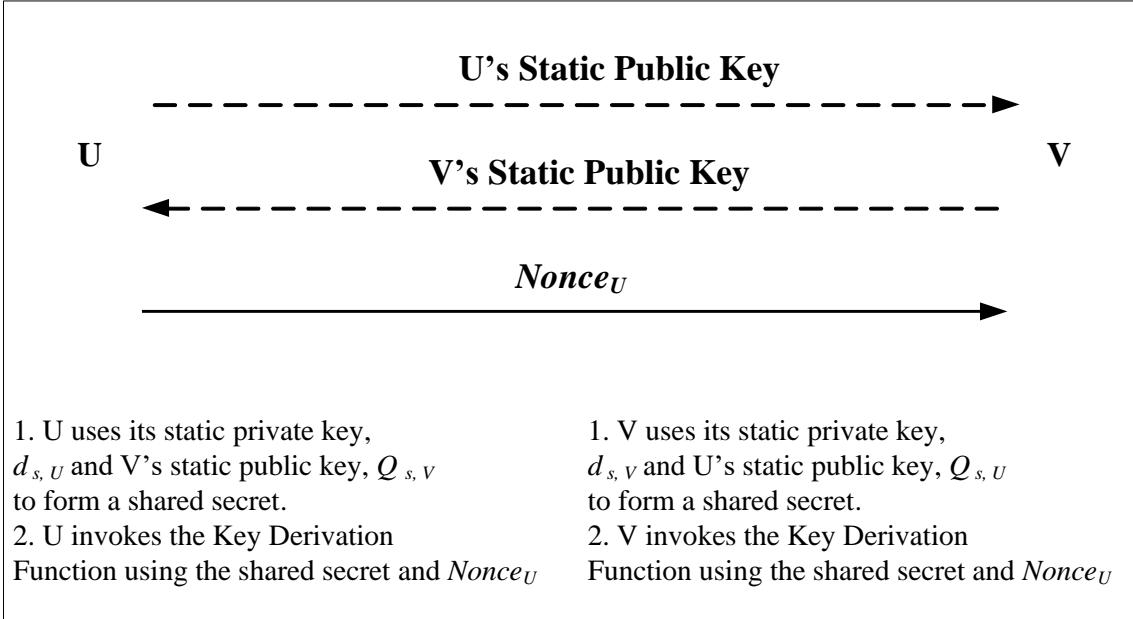
This scheme is for use by a DLMS server and another party to agree on an ephemeral encryption key to protect xDLMS APDUs or COSEM data. The party sending the message (the originator) plays the role of party U and the other party (the recipient) plays the role of party V.

NOTE 1 The terms originator and recipient refer to fields of the general-ciphering APDU.

In this case, the parties use only static key pairs. Each party obtains the other party's static public key. A nonce,  $Nonce_u$ , is sent by party U to party V to ensure that the derived keying material is different for each key-establishment transaction.

The parties derive the shared secret Z using their own static private key and the other party's static public key. Secret keying material is derived using the key-derivation function specified in 9.2.3.4.6.5 the shared secret Z, U and V's identifier and the nonce.

The process is shown in Figure 99 and Table 24.



NOTE This figure is based on NIST SP 800-56A Rev. 2: 2013, Figure 15.

**Figure 99 – C(0e, 2s) scheme: each party contributes only a static key pair**

Prerequisites:

- a) each party shall have an authentic copy of the same set of domain parameters,  $D$ .  $D$  must be selected from one of the two sets of domain parameters, see Annex A;
- b) each party has been designated as the owner of a static key pair that was generated as specified in 9.2.6.2 using the set of domain parameters,  $D$ ;
- c) the parties have agreed on using the NIST Concatenation KDF, see 9.2.3.4.6.5. SHA-256 is the hash function to use with the domain parameters for P-256 and SHA-384 is the hash function to use with the domain parameters for P-384;
- d) prior to or during the key agreement process, each party shall obtain the identifier associated with the other party during the key agreement scheme;
- e) both parties shall obtain the static public key of the other party that is bound to the identifier. These static public keys shall be obtained in a trusted manner (e.g., from a certificate signed by a trusted CA).

NOTE 2 See also NIST SP 800-56A Rev. 2: 2013 for additional information on assurance on the validity of the domain parameters, the validity of the private and public key and the assurance of the possession of the private key.

**Table 24 – Static Unified Model key agreement scheme summary**

	<b>Party U</b>	<b>Party V</b>
Domain Parameters	( $q, FR, a, b\{SEED\}, G, n, h$ ) As determined by the security suite	( $q, FR, a, b\{SEED\}, G, n, h$ ) As determined by the security suite
Static Data	Static private key, $d_{s, U}$ Static public key, $Q_{s, U}$	Static private key, $d_{s, V}$ Static public key, $Q_{s, V}$
Ephemeral Data	Nonce <sub>U</sub>	
Computation	Compute $Z$ by calling ECC CDH using $d_{s, U}$ and $Q_{s, V}$	Compute $Z$ by calling ECC CDH using $d_{s, V}$ and $Q_{s, U}$
Derive Secret Keying Material	<ol style="list-style-type: none"> <li>1. Compute DerivedKeyingMaterial using Nonce<sub>U</sub></li> <li>2. Destroy <math>Z</math></li> </ol>	<ol style="list-style-type: none"> <li>1. Compute DerivedKeyingMaterial using Nonce<sub>U</sub></li> <li>2. Destroy <math>Z</math></li> </ol>
NOTE 1 This table is based on NIST SP 800-56A Rev. 2: 2013, Table 26.		
NOTE 2 The value of $Z$ is the same in all C(0e, 2s) key-establishment transactions between the same two parties, therefore if it is ever compromised, then all of the keying material derived in past, current, and future C(0e, 2s) key-agreement transactions between these same two entities that employ these same static key pairs may be compromised as well. Any shared secret $Z$ that is not 'zeroized' shall be stored and used with the same security protections as private keys.		

The rationale for choosing this scheme is specified in NIST SP 800-56A Rev. 2: 2013, 8.5.

The use of this scheme in DLMS/COSEM is further explained in C.3.

#### 9.2.3.4.6.5     Key Derivation Function – The NIST Concatenation KDF

The NIST Concatenation KDF as specified in NIST SP 800-56A Rev. 2: 2013, 5.8.1.1 and NSA2, Clause 5 shall be used. It is as follows:

**Function call:** kdf( $Z, OtherInput$ )

where  $OtherInput$  consists of  $keydatalen$  and  $OtherInfo$ .

Implementation-Dependent Parameters:

- a)  $hashlen$ : an integer that indicates the length (in bits) of the output of the hash function,  $hash$ , used to derive blocks of secret keying material. This will be 256 (for SHA-256) in the case of security suite 1 and 384 (for SHA-384) in the case of security suite 2;
- b)  $max\_hash\_inputlen$ : an integer that indicates the maximum length (in bits) of the bit string(s) input to the hash function.

The length shall be less than  $2^{64}$  bits for SHA-256 and less than  $2^{128}$  bits for SHA-384.

**Function:** H: a hash function: SHA-256 in the case of security suite 1 and SHA-384 in the case of security suite 2.

Input:

- c)  $Z$ : a byte string that represents the shared secret  $z$ ;
- d)  $keydatalen$ : an integer that indicates the length (in bits) of the secret keying material to be generated: 128 bit for security suite 1 and 256 bit for security suite 2;
- e)  $OtherInfo$ : A bit string equal to the following concatenation:

*AlgorithmID // PartyUInfo // PartyVInfo {||SuppPubInfo}{||SuppPrivInfo}*

where the subfields are defined as follows:

- i)  $AlgorithmID$ : A bit string that indicates how the derived secret keying material will be parsed and for which algorithm(s) the derived secret keying material will be used. When key

agreement is used to agree on the GUEK and GAK, then the algorithm ID used shall be AES-GCM-128 / AES-GCM-256. When it is used to agree on the KEK then the algorithm ID used shall be AES-WRAP-128 / AES-WRAP-256. See Table 26;

- ii) *PartyUInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by Party U to the key derivation process;
- iii) *PartyVInfo*: A bit string containing public information that is required by the application using this KDF to be contributed by Party V to the key derivation process;
- iv) (Optional) *SuppPubInfo*: A bit string containing additional, mutually known public information. Not used in DLMS/COSEM;
- v) (Optional) *SuppPrivInfo*: A bit string containing additional, mutually known private information (for example, a shared secret symmetric key that has been communicated through a separate channel). Not used in DLMS/COSEM.

The format and content of each subfield and substring is specified in Table 25.

**Table 25 – *OtherInfo* subfields and substrings**

Subfield Substring	Key agreement scheme			Length in octets	Value
	C(2e, 0s)	C(1e, 1s)	C(0e, 2s)		
	Format				
<i>AlgorithmID</i>	Fixed	Fixed	Fixed	7	See Table 26
<i>PartyUInfo</i>	Fixed	Fixed	Variable	8+n	–
<i>ID<sub>U</sub></i>	Fixed	Fixed	Fixed	8	originator-system-title
<i>NonceU</i>	–	–	Variable	n	<i>DataLen</i> = length of transaction-id, shall be 1 octet <i>Data</i> = value of transaction-id
<i>PartyVInfo</i>	Fixed	Fixed	Fixed	8	–
<i>ID<sub>V</sub></i>	Fixed	Fixed	Fixed	8	recipient-system-title

“originator-system-title”, “transaction-id” and “recipient-system-title” are the fields of the general-ciphering APDU

In DLMS/COSEM, key derivation delivers a single key for a given purpose. The length is as determined by the security suite. *AlgorithmId* shall be as specified in Table 26. See also 9.4.2.2.4.

**Table 26 – Cryptographic algorithm ID-s**

Algorithm	COSEM cryptographic algorithm ID	Encoded value
AES-GCM-128	2.16.756.5.8.3.0	60 85 74 05 08 03 00
AES-GCM-256	2.16.756.5.8.3.1	60 85 74 05 08 03 01
AES-WRAP-128	2.16.756.5.8.3.2	60 85 74 05 08 03 02
AES-WRAP-256	2.16.756.5.8.3.3	60 85 74 05 08 03 03

NOTE Attention is drawn to a correction to this table where the encoded values were erroneous in versions of the Green Book prior to Ed. 8.2.

### 9.2.3.5 Random number generation

Strong random number generator (RNG) shall be provided to generate the random numbers required for the various algorithms used in DLMS/COSEM. The RNG shall be preferably non-deterministic. If a non-deterministic RBG is not available, the system shall make use of sufficient entropy to create a good quality seed for a deterministic RNG.

### 9.2.3.6 Compression

NOTE Compression does not involve cryptography, but it is a transformation of the xDLMS APDU. For this reason, and as it is controlled together with symmetric key ciphering it is specified here.

226/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

Compression can be applied to COSEM data or xDLMS APDUs. This process can be combined with symmetric key ciphering. See 9.2.7.2.4.7 and 9.2.7.2.4.8. The compression algorithm shall be as specified in ITU-T V.44: 2000 with the packet method specified in ITU-T V.44: 2000, Annex B.1. This algorithm has been selected for to meet the following requirements:

- low processing load;
- low memory requirements;
- low latency.

The use of compression is indicated by bit 7 of the Security Control Byte, specified in 9.2.7.2.4.

#### 9.2.3.7 Security suite

A security suite determines the set of cryptographic algorithms available for the various cryptographic primitives and the key sizes.

The DLMS/COSEM security suites – see Table 27 – are based on NSA Suite B and include cryptographic algorithms for authentication, encryption, key agreement, digital signature and hashing specifically:

- authentication and encryption: the Advanced Encryption Standard (AES) shall be used as specified in FIPS PUB 197, with key sizes of 128 and 256 bits. AES shall be used with the Galois/Counter Mode (GCM) of operation specified in NIST SP 800-38D:2007;
- digital signature: the Elliptic Curve Digital Signature Algorithm (ECDSA) shall be used as specified FIPS PUB 186-4:2013 and in NSA1, using the curves P-256 or P-384;
- key agreement:
  - the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme, see 9.2.3.4.6.2;
  - the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme, see 9.2.3.4.6.3; and
  - the Static Unified Model C(0e, 2s, ECC CDH) scheme, see 9.2.3.4.6.4 shall be used using the elliptic curves P-256 or P-384.
- hashing: the Secure Hash Algorithms (SHA) SHA-256 and SHA-384 shall be used as specified in FIPS PUB 180-4:2012.

In addition, a key wrapping and a compression algorithm are available.

**Table 27 – DLMS/COSEM security suites**

Security Suite Id	Security suite name	Authenticated encryption	Digital signature	Key agreement	Hash	Key-transport	Compression
0	AES-GCM-128	AES-GCM-128	–	–	–	AES-128 key wrap	–
1	ECDH-ECDSA-AES-GCM-128-SHA-256	AES-GCM-128	ECDSA with P-256	ECDH with P-256	SHA-256	AES-128 key wrap	V.44
2	ECDH-ECDSA-AES-GCM-256-SHA-384	AES-GCM-256	ECDSA with P-384	ECDH with P-384	SHA-384	AES-256 key wrap	V.44
All other reserved	–	–	–	–	–	–	–

#### 9.2.4 Cryptographic keys – overview

A cryptographic key is a parameter used in conjunction with a cryptographic algorithm that determines its operation in such a way that an entity with knowledge of the key can reproduce or reverse the operation, while an entity without knowledge of the key cannot. In DLMS/COSEM, examples of operations include:

- the transformation of plaintext into ciphertext;
- the transformation of ciphertext into plaintext;
- the computation and verification of an authentication code (MAC);
- key wrapping;

- applying and verifying digital signature;
- key agreement.

Keys used with symmetric key algorithms are specified in 9.2.5.

Keys used with public key algorithms are specified in 9.2.6.

### **9.2.5 Key used with symmetric key algorithms**

#### **9.2.5.1 Symmetric keys types**

Symmetric keys are classified according to:

a) their purpose:

- 1) a key encrypting key (KEK) is used to encrypt / decrypt other symmetric keys; see 9.2.5.4. In DLMS/COSEM this is the master key;
- 2) an encryption key is used as the block cipher key of the AES-GCM algorithm, see also 9.2.3.3.7.4;
- 3) an authentication key is used as Additional Authenticated Data (AAD) in the AES-GCM algorithm, see also 9.2.3.3.7.5.

b) their lifetime:

- 1) static keys that are intended to be used for a relatively long period of time. In DLMS/COSEM these may be:
  - a global key that may be used over several AAs established repeatedly between the same partners. A global key may be a unicast encryption key (GUEK), a broadcast encryption key (GBEK) or an authentication key (GAK);
  - a dedicated key that may be used repeatedly during a single AA established between two partners. Therefore, its lifetime is the same as the lifetime of the AA. A dedicated key can be only a unicast encryption key.
- 2) ephemeral keys used generally for a single exchange within an AA.

For generation and distribution of symmetric keys, see NIST SP 800-57:2012, 8.1.5.2.

A master key and global keys are established between each DLMS client – server pair using one of the methods shown in Table 28. They should be renewed in appropriate intervals, see 9.2.3.3.7.3 and 9.2.5.6.

Dedicated keys are generated by the DLMS client and transported to the server in the dedicated-key field of the xDLMS InitiateRequest APDU, carried by the user-information field of the AARQ APDU. When the dedicated key is present, the xDLMS InitiateRequest APDU shall be authenticated and encrypted using the AES-GCM-128 / 256 algorithm, the global unicast encryption key and – if in use, see 9.2.3.3.7.5 – the authentication key. The xDLMS InitiateResponse APDU, carried by the user-information field of the AARE APDU shall be also encrypted and authenticated the same way. When the dedicated key is used, the key-set bit of the security control byte, see Table 45 – is not relevant and shall be set to zero.

NOTE The AARQ and the AARE APDUs themselves are not protected.

Table 28 summarizes the symmetric key types, their purpose, the methods to establish them and their use with the different APDUs and between different entities.

**Table 28 – Symmetric keys types**

Key type	Purpose	Key establishment	Use
Master key, KEK <sup>1)</sup>	Key Encrypting Key (KEK) for : - (new) master key; - global encryption or authentication keys; - ephemeral encryption keys.	Out of band	Can be identified as the KEK in general-ciphering APDUs between client-server, see Table 29
		Key wrapping	
		Key agreement <sup>3)</sup>	
Global unicast encryption key, GUEK <sup>2)</sup>	Block cipher key for unicast - xDLMS APDUs and / or - COSEM Data	Key wrapping	- service-specific global ciphering APDU client-server - general-glo-ciphering APDU client-server - general-ciphering APDU client-server - "Data protection" object protection parameters
		Key agreement <sup>3)</sup>	
Global broadcast encryption key, GBEK <sup>2)</sup>	Block cipher key for broadcast - xDLMS APDUs and / or - COSEM Data	Key wrapping	- "Data protection" object protection parameters
(Global) Authentication key, GAK <sup>2)</sup>	Part of AAD to the ciphering process of xDLMS APDUs and / or COSEM data	Key wrapping	All APDUs between client-server and third party-server
		Key agreement <sup>3)</sup>	
Dedicated key (unicast)	Block cipher key of unicast xDLMS APDUs, within an established AA	Key-transport in xDLMS Initiate.request APDU	- service-specific dedicated ciphering APDU client-server - general-ded-ciphering APDU client-server during the lifetime of an AA
Ephemeral encryption key	Block cipher key for: - xDLMS APDUs and/or - COSEM data	Key wrapping	- general-ciphering APDU client-server - "Data protection" object protection parameters
	Block cipher key for: - xDLMS APDUs and/or - COSEM data	Key agreement <sup>4)</sup>	- general-ciphering APDU client-server or third-party server - "Data protection" object protection parameters

1) Held by a "Security setup" object. Different AAs may use the same or different "Security setup" objects.  
 2) Held by a "Security setup" object. Different AAs may use the same or different "Security setup" objects. The use of the GUEK or the GBEK can be identified by:  
    – the key-set bit of the Security Control Byte, see Table 45; or  
    – by the key-id parameter of the general-ciphering APDU, see Table 29;  
    – or by the protection parameters of the "Data protection" IC, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.9.  
 3) Established using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme, see 9.2.3.4.6.2  
 4) Established using the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme or the Static Unified Model C(0e, 2s, ECC CDH) scheme. See 9.2.3.4.6.3 and 9.2.3.4.6.4.

### 9.2.5.2 Key information with general-ciphering APDU and data protection

When the general-ciphering APDU is used to protect xDLMS APDUs or when COSEM data is protected, the sender sends the necessary information on the key that has been / shall be used to cipher / decipher the xDLMS APDU / COSEM data, together with the ciphered xDLMS APDU / COSEM data.

The key information required is summarized in Table 29 and further specified in 9.2.5.3, 9.2.5.4 and 9.2.5.5.

**Table 29 – Key information with general-ciphering APDU and data protection**

Key information choices		Comment
<b>key-Info</b>		
<b>identified-key</b>	S	The EK is identified
key-id	M	
global-unicast-encryption-key	S	GUEK
global-broadcast-encryption-key	S	GBEK
<b>wrapped-key</b>	S	The EK is transported using key wrapping
kek-id	M	
master-key	M	Identifies the key used for wrapping the key-ciphered-data. 0 = Master Key (KEK)
key-ciphered-data	M	Randomly generated key wrapped with KEK
<b>agreed-key</b>	S	The key is agreed by the parties using either: - the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme see 9.2.3.4.6.3; or - the Static Unified Model C(0e, 2s, ECC CDH) scheme see 9.2.3.4.6.4.
key-parameters	M	Identifier of the key agreement scheme: 0x01: C(1e, 1s ECC CDH) 0x02: C(0e,2s ECC CDH) All other reserved.
key-ciphered-data	M	- In the case of the C(1e, 1s, ECC CDH) scheme: the public key $Q_{e, U}$ , of the ephemeral key agreement key pair of party U, signed with the private digital signature key of party U. - In the case of the C(0e, 2s, ECC CDH) scheme: an octet-string of length zero. In this case party U has to provide a nonce, $Nonce_U$ . See 9.2.3.4.6.4 and 9.2.3.4.6.5.
<p>M: Mandatory (part of a SEQUENCE)  S: Selectable (part of a CHOICE)  For the ASN.1 specification, see 9.5.</p> <p>NOTE Using key identification restricts exchanging protected xDLMS APDUs / COSEM data between a client and a server because the GUEK and the GBEK shall not be disclosed to any party other than the client and the server.</p>		

### 9.2.5.3 Key identification

The key identified may be the Global Unicast Encryption Key (GUEK) or the Global Broadcast Encryption Key (GBEK). In this case, the key-set bit of the security control byte – see Table 45 – is not relevant and shall be set to zero.

### 9.2.5.4 Key wrapping

Key wrapping can be used to establish static or ephemeral symmetric keys.

The algorithm is the AES key wrap algorithm specified in 9.2.3.3.8. The KEK is the master key. Consequently, this method can be used only between parties sharing the master key, i.e. between a client and a server.

The static keys that can be established using key wrapping may be:

- the master key, KEK; and/or
- the global unicast encryption key GUEK; and/or
- the global broadcast encryption key GBEK; and/or
- the (global) authentication key, GAK.

To establish these static keys using key wrap, the key shall be first generated by the client, then it shall be transferred to the server by invoking the *key\_transfer* method of the “Security setup” object, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7. The method invocation parameter shall carry the key\_id(s) and the wrapped key(s). The APDU carrying the service that invokes the method and the method invocation parameters shall be protected as required by the security policy and the access rights.

NOTE The required level of protection can be specified in project specific companion specifications.

To establish an ephemeral key using key wrapping, the originator of the xDLMS APDU or the COSEM data randomly generates an ephemeral key. This key shall be wrapped using the AES key wrap algorithm and the KEK and shall be sent to the recipient together with the xDLMS APDU or the COSEM data that have been ciphered using the ephemeral key. The recipient shall unwrap the key then it shall use it to decipher the xDLMS APDU / COSEM data received.

#### 9.2.5.5 Key agreement

Key agreement can be used to establish static keys between a server and a client or ephemeral keys between a server and a client or a third party. Different key agreement schemes are available to establish different keys.

The Ephemeral Unified Model C(2e,0s, ECC CDH) scheme can be used by the client and the server to agree on the:

- master key, KEK; and/or
- global unicast encryption key GUEK; and/or
- global broadcast encryption key GBEK; and/or
- (global) authentication key, GAK.

This scheme is supported by the *key\_agreement* method of the “Security setup” interface class, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7. The method invocation parameters carry the necessary parameters as specified in 9.2.3.4.6.2. The APDUs carrying the service that invokes the method as well as the method invocation parameters shall be protected as required by the security policy and the access rights. See also Annex C.

NOTE The required level of protection can be specified in project specific companion specifications.

To establish an ephemeral encryption key – used as the block cipher key – using key agreement, two schemes are available:

- the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme specified in 9.2.3.4.6.3;  
Unless specified otherwise in a project specific companion specification, the C(1e, 1s ECC CDH) scheme shall be used.
- the Static Unified Model C(0e, 2s, ECC CDH) scheme specified in 9.2.3.4.6.4.

#### 9.2.5.6 Symmetric key cryptoperiods

Symmetric key cryptoperiods should be determined in project specific companion specifications. Recommendations are given in NIST SP 800-57:2012, Part 1, 5.3.5 *Symmetric Key Usage Periods and Cryptoperiods* and 5.3.6 *Cryptoperiod Recommendations for Specific Key Types*.

### 9.2.6 Keys used with public key algorithms

#### 9.2.6.1 Overview

Asymmetric keys – see Table 30 – are classified according to:

- their purpose: digital signature key or key agreement key;
- by their lifetime: static keys or ephemeral keys.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	231/614
-----------------------	------------	-----------------------	---------

**Table 30 – Asymmetric keys types and their use**

Digital signature			
Key type	Signatory	Verifier	Use
Digital signature key pair	Private key	Public key	<p>Signatory uses private key to compute digital signature:</p> <ul style="list-style-type: none"> <li>- on an xDLMS APDUs; and/or</li> <li>- on COSEM data; or</li> <li>- on an ephemeral public key agreement key.</li> </ul> <p>Verifier uses public key to verify digital signature</p> <ul style="list-style-type: none"> <li>- on an xDLMS APDUs; and/or</li> <li>- on COSEM data; or</li> <li>- on an ephemeral public key agreement key received.</li> </ul>
Key agreement			
Key type	Party U	Party V	Use
Ephemeral key agreement key pair	Private key Public key	Private key Public key	<p>In the case of the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme both parties have an ephemeral key pair. See 9.2.3.4.6.2.</p> <p>In the case of the One-Pass Diffie-Hellman C(1e,1s, ECC CDH) scheme only party U has an ephemeral key pair. See 9.2.3.4.6.3</p>
Static key agreement key pair	Private key Public key	Private key Public key	<p>In the case of the One-Pass Diffie-Hellman C(1e,1s, ECC CDH) scheme only party V has a static key pair. See 9.2.3.4.6.3.</p> <p>In the case of the Static Unified Model, C(0e, 2s, ECC CDH) scheme both the party U and party V have a static key pair. See 9.2.3.4.6.4.</p>

### 9.2.6.2 Key pair generation

An ECC key pair  $d$  and  $Q$  is generated for a set of domain parameters  $(q, FR, a, b \{, domain\_parameter\_seed\}, G, n, h)$ . Two methods are provided for the generation of the ECC private key  $d$  and public key  $Q$ ; one of these two methods shall be used to generate  $d$  and  $Q$ .

Prior to generating ECDSA key pairs, assurance of the validity of the domain parameters  $(q, FR, a, b \{, domain\_parameter\_seed\}, G, n, h)$  shall have been obtained.

For details, see FIPS PUB 186-4:2013, Annex B.4.

### 9.2.6.3 Public key certificates and infrastructure

#### 9.2.6.3.1 Overview

This subclause 9.2.6.3 describes the public key certificates for the purposes of DLMS/COSEM and an example PKI infrastructure to manage them. It is based on the following documents:

- NIST SP 800-21:2005 and NIST SP 800-32:2001, providing a general description of public key cryptography and public key infrastructures;
- ITU-T X.509:2008, specifying public key and attribute certificate frameworks;
- RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile;
- NSA3 specifying the NSA Suite B Base Certificate and CRL Profile.

The trust model is described in 9.2.6.3.2.

A PKI architecture – as an example – is described in 9.2.6.3.3.

The certificate and certificate extension profile is specified in 9.2.6.4.

The public key certificates to be held by DLMS servers are specified in 9.2.6.5.

The management of certificates is specified in 9.2.6.6.

#### 9.2.6.3.2 Trust model

For DLMS/COSEM based device data exchange systems using public key cryptography various public-private key pairs and public key certificates should be in place.

A public key certificate binds a public key to an identity: the subject. A certificate is digitally signed by a Certification Authority.

To provide and manage the certificates, some form of Public Key Infrastructure is required. A PKI consists of Certification Authorities issuing certificates and end entities using these certificates. For a PKI example, see 9.2.6.3.3.

In its simplest form, a certification hierarchy consists of a single CA. However, the hierarchy usually contains multiple CAs that have clearly defined parent-child relationships. It is also possible to deploy multiple hierarchies.

The PKI needs a trust anchor that is used to validate the first certificate in a sequence of certificates. The trust anchor may be a Root-CA certificate, a Sub-CA certificate or a directly trusted key.

NOTE 1 Trust anchors are Certificates or directly trusted keys marked as such. However, this marking is out of the Scope of this Technical Report.

DLMS servers shall be provisioned with one or more trust anchors during manufacturing using a trusted Out of Band (OOB) process.

NOTE 2 Provisioning clients and third parties with trust anchors is out of the Scope of this Technical Report.

DLMS servers may also be provisioned with their own certificates and certificates of CAs, DLMS clients and third parties. This may also happen using a trusted OOB process or through the “Security setup” object.

The “Security setup” interface class – see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7 – provides:

- an attribute that provides information on the certificates stored on the server;
- a method to generate server key pairs and a method to generate Certificate Signing Request (CSR) information on the server to be sent by the client to a CA;
- methods to import, export and remove certificates.

These attributes / methods can be used to manage the certificates by the client or by third parties via the client acting as a broker. Messages accessing the attributes and methods of “Security setup” objects and the data included shall be suitably protected.

Certificates generally have a validity period. However, certificates issued to DLMS servers may be indefinitely valid. Certificates may be replaced when they expire.

Before a server uses a Certificate, it has to be verified. Verification includes:

- checking syntactic validity of the certificate;
- checking the attributes included in the certificate;
- checking that the certificate validity period has not expired;
- checking the certification path to the trust anchor;
- checking the signature of the issuer of the certificate.

It is assumed that the trust anchor, other CA-certificates, as well as the certificates of DLMS clients and third parties held by the server are all valid. It is the responsibility of the system to replace / remove any certificates the validity of which have expired or that have been revoked.

Clients and third parties also have to verify server certificates before using them. They may have capabilities to verify the status of certificates they are using. However this is outside the Scope of this Technical Report.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	233/614
-----------------------	------------	-----------------------	---------

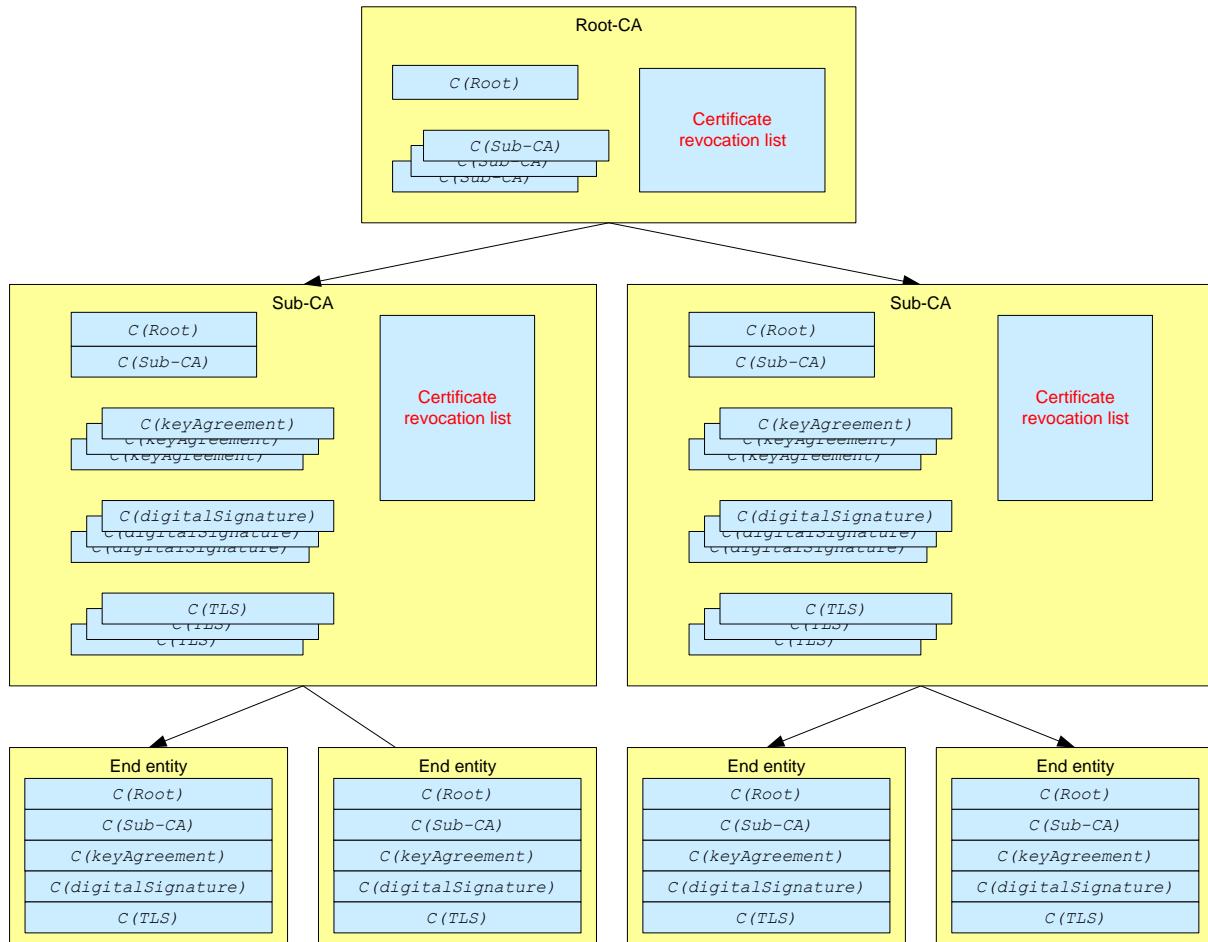
### 9.2.6.3.3 PKI architecture – informative

#### 9.2.6.3.3.1 General

NOTE 1 The introductory text is quoted from NIST SP 800-21:2005, 3.7.

A PKI is a security infrastructure that creates and manages public key certificates to facilitate the use of public key (i.e., asymmetric key) cryptography. To achieve this goal, a PKI needs to perform two basic tasks:

- 1) Generate and distribute public key certificates to bind public keys to other information after validating the accuracy of the binding; and
- 2) Maintain and distribute certificate status information for unexpired certificates.



**Figure 100 – Architecture of a Public Key Infrastructure (example)**

For DLMS/COSEM a hierarchical PKI is foreseen, comprising the following components as shown in Figure 100.

NOTE 2 The actual structure of the PKI is left to project specific companion specifications to meet the operators' needs.

- Root Certification Authority (Root-CA): see 9.2.6.3.3.2;
- Certification Authority / Subordinate Authority (Sub-CA): see 9.2.6.3.3.3;
- End entities: entities that do not issue certificates: DLMS clients, DLMS servers and third parties: see 9.2.6.3.3.4.

#### 9.2.6.3.3.2 Root-CA

The Root-CA provides the trust anchor of the PKI. It issues certificates for Sub-CAs and maintains a certificate revocation list (CRL). The Root-CA Certificate Policy defines the rules for handling the issuance of certificates.

The Root-CA owns the Root certificate  $C(\text{Root})$ . The Certificate of the Root-CA is self-signed with the Root-CA private key. Sub-CA certificates are also signed with the Root-CA private key.

#### 9.2.6.3.3.3 Sub-CA

A Sub-CA is an organization that issues certificates for end entities. Each Sub-CA is authorised by the Root-CA to do so.

NOTE Sub-CAs may be independent organizations, or may be market participants, device operators, manufacturers.

Each Sub-CA must handle a Certificate Policy, which must comply with the Root-CA Certificate Policy.

Sub-CAs also maintain the list of certificates issued to end entities and a Certification Revocation List.

A sub-CA owns a certificate  $C(\text{Sub-CA})$ . This certificate is issued by the Root-CA. The private key of the Sub-CA is used for signing end entity certificates.

#### 9.2.6.3.4 End entities

In the PKI infrastructure, an end entity is a user of PKI certificates and/or an end user system that is the subject of a certificate. The following end entity certificates are defined:

- digital signature key certificate  $C(\text{digitalSignature})$ , used for digital signature;
- static key agreement key certificate  $C(\text{keyAgreement})$ , used for key agreement;
- [optional] TLS-Certificate  $C(\text{TLS})$ , used for performing authentication between a DLMS client and a DLMS server prior the establishment of a TLS secure channel.

See also 9.2.6.5.

### 9.2.6.4 Certificate and certificate extension profile

#### 9.2.6.4.1 General

This subclause 9.2.6.4 specifies the certificate and certification extension profile as required for DLMS/COSEM systems using public key cryptography.

All certificates shall have the structure specified for X.509 version 3 certificates.

For the tables presenting the fields of the certificate and certificate extensions, the following notation applies:

- m (mandatory): the field must be filled;
- o (optional): the field is optional;
- x (do not use): the field shall not be used.

Each certificate extension is designated either as critical or non-critical. A certificate-using system MUST reject the certificate if it encounters a critical extension it does not recognize or a critical extension that contains information that it cannot process. A non-critical extension MAY be ignored if it is not recognized, but MUST be processed if it is recognized.

This Technical Report specifies minimum requirements. Project specific companion specifications may specify more strict requirements, for example a field specified in this Technical Report as optional may be made mandatory or an extension designated as non-critical may be designated as critical.

#### 9.2.6.4.2 The X.509 v3 Certificate

An X.509 v3 certificate is a SEQUENCE of three required fields as shown in Table 31.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	235/614
-----------------------	------------	-----------------------	---------

**Table 31 – X.509 v3 Certificate structure**

Certificate field	RFC 5280 reference	m/x/o	Value
Certificate	4.1.1		
tbsCertificate	4.1.1.1	m	See below
signatureAlgorithm	4.1.1.2	m	See below
signatureValue	4.1.1.3	m	See below

The tbsCertificate field contains the names of the subject and issuer, a public key associated with the subject, a validity period, and other associated information. The fields of the tbsCertificate are summarized in Table 32. The tbsCertificate usually includes extensions; these are described in 9.2.6.4.4.

The signatureAlgorithm field contains the identifier for the cryptographic algorithm used by the CA to sign this certificate.

```
AlgorithmIdentifier ::= SEQUENCE {
    algorithm           OBJECT IDENTIFIER
    parameters          ANY DEFINED BY algorithm OPTIONAL }
```

The two algorithm identifiers used in DLMS/COSEM are:

- ecdsa-with-SHA256, OID 1.2.840.10045.4.3.2 in the case of security suite 1;
- ecdsa-with-SHA384, OID 1.2.840.10045.4.3.3 in the case of security suite 2;

The signatureValue contains a digital signature computed upon the ASN.1 DER encoded tbsCertificate. The ASN.1 DER encoded tbsCertificate is used as the input to the signature function.

By generating this signature, a CA certifies the validity of the information in the tbsCertificate field. In particular, the CA certifies the binding between the public key material and the subject of the certificate.

### 9.2.6.4.3 tbsCertificate

#### 9.2.6.4.3.1 Overview

The fields of the tbsCertificate are shown in Table 32.

**Table 32 – X.509 v3 tbsCertificate fields**

Certificate field	RFC 5280 reference	Clause	m/x/o	Comment
tbsCertificate	4.1.2	9.2.6.4.2		
Version	4.1.2.1	–	m	'v3' (value is 2)
Serial Number	4.1.2.2	9.2.6.4.3.2	m	Certificate serial number assigned by the CA (not longer than 20 octets)
Signature	4.1.2.3	–	m	Same algorithm identifier as the signatureAlgorithm in the Certificate
Issuer	4.1.2.4	9.2.6.4.3.3	m	Distinguished name (DN) of the certificate issuer.
Validity	4.1.2.5	9.2.6.4.3.4	m	Validity of the certificate.
Subject	4.1.2.6	9.2.6.4.3.3	m	Distinguished name (DN) of the certificate subject.
Subject Public Key Info	4.1.2.7	9.2.6.4.3.5	m	
Issuer Unique ID	4.1.2.8		x	Not applicable

Certificate field	RFC 5280 reference	Clause	m/x/o	Comment
Subject Unique ID	4.1.2.8	9.2.6.4.3.6	o	
Extensions	4.1.2.9	9.2.6.4.4	m	

#### 9.2.6.4.3.2 Serial number

As specified in RFC 5280, 4.1.2.2 the serial number MUST be a positive integer assigned by the CA to each certificate. It MUST be unique for each certificate issued by a given CA (i.e., the issuer name and serial number identify a unique certificate).

Certificate users MUST be able to handle `serialNumber` values up to 20 octets. Conforming CAs MUST NOT use `serialNumber` values longer than 20 octets.

#### 9.2.6.4.3.3 Issuer and Subject

The `Issuer` field identifies the entity that has signed and issued the certificate.

The `Subject` field identifies the entity associated with the public key stored in the `subject` public key field. The subject name MAY be carried in the `Subject` field and/or the `subjectAltName` extension. If subject naming information is present only in the `subjectAltName` extension then the subject name MUST be an empty sequence and the `subjectAltName` extension MUST be critical. See 9.2.6.4.4.6.

The naming scheme of the various entities of the PKI is shown in the following tables. The names shall be inserted in the `Issuer` or the `Subject` field of the `tbsCertificate` as applicable. See Table 33, Table 34 and Table 35.

**Table 33 – Naming scheme for the Root-CA instance (informative)**

Attribute	Abbreviation	m/x/o	Name	Comment
Common Name	CN	m	<Root-CA>	Name of Root-CA
Organization	O	o	<PKI-Name>	Name of PKI
Organizational Unit	OU	o		Name of organizational unit
Country	C	o		ISO 3166 country code
NOTE Values within the less-than – greater-than signs "< >" are to be assigned by the PKI or the CA as applicable.				

**Table 34 – Naming scheme for the Sub-CA instance (informative)**

Attribute	Abbrev.	m/x/o	Name	Comment
Common Name	CN	m	<XXX-CA>	Name of sub-CA The CN shall finish with "CA" so that the CA function is recognized.
Organization	O	o	<PKI-Name>	Name of PKI
Organizational Unit	OU	o		Name of organizational unit
Country	C	o		ISO 3166 country code
Locality	L	o	<Locality>	Locality where the Sub-CA is located
State	ST	o	<State>	
NOTE Values within the less-than – greater-than signs "< >" are to be assigned by the PKI or the CA as applicable.				

The format of the elements of the naming scheme of Root-CA and Sub-CA instances is left to project specific companion specifications.

**Table 35 – Naming scheme for the end entity instance**

Attribute	Abbrev.	m/x/o	Name	Comment
Common Name	CN	m	<System-title>	DLMS/COSEM System title: 8 bytes represented as a 16 characters: Example: "4D4D4D0000BC614E"
Organization	O	o	<PKI-Name>	Name of PKI
Organizational Unit	OU	o		Name of organizational unit
Country	C	o		ISO 3166 country code
Locality	L	x	<Locality>	Locality where the entity is located
State	S	x	<State>	

NOTE Values within the less-than – greater-than signs "< >" are to be assigned by the PKI or the CA as applicable.

#### 9.2.6.4.3.4 Validity period

The certificate validity period is the time interval during which the CA warrants that it will maintain information about the status of the certificate. The field is represented as a SEQUENCE of two dates:

- the date on which the certificate validity period begins (`notBefore`); and
- the date on which the certificate validity period ends (`notAfter`).

In the case of CA certificates, Sub-CA certificates and end-entities other than DLMS servers `notBefore` and `notAfter` shall be well defined dates.

DLMS servers may be given certificates for which no good expiration date can be assigned; such a certificate is intended to be used for the entire lifetime of the device.

To indicate that a certificate has no well-defined expiration date, the `notAfter` SHOULD be assigned the GeneralizedTime value of 99991231235959Z.

For details, see RFC 5280, 4.1.2.5.

#### 9.2.6.4.3.5 SubjectPublicKeyInfo

The field `SubjectPublicKeyInfo` shall have the following structure:

```
SubjectPublicKeyInfo ::= SEQUENCE {
    Algorithm                  AlgorithmIdentifier,
    subjectPublicKey           BIT STRING }
```

An algorithm identifier is defined by the following ASN.1 structure:

```
AlgorithmIdentifier ::= SEQUENCE {
    algorithm                OBJECT IDENTIFIER,
    parameters               ANY DEFINED BY algorithm OPTIONAL }
```

The algorithm identifier is used to identify a cryptographic algorithm. The OBJECT IDENTIFIER component identifies the algorithm (such as DSA with SHA-1). The contents of the optional parameters field will vary according to the algorithm identified. This field MUST contain the same algorithm identifier as the signature field in the sequence `tbsCertificate`; see 9.2.6.4.2.

The algorithm OBJECT IDENTIFIER shall contain the value

- OID value: 1.2.840.10045.2.1;
- OID description: ECDSA and ECDH Public Key.

The value of the parameter shall be 1.2.840.10045.3.1.7 for the curve NIST P-256 and 1.3.132.0.34 for the curve NIST P-384.

The subjectPublicKey from SubjectPublicKeyInfo is the ECC public key. ECC public keys have the following syntax:

ECPoint ::= OCTET STRING

Implementations of Elliptic Curve Cryptography according to this Technical Report MUST support the uncompressed form and MAY support the compressed form of the ECC public key. The hybrid form of the ECC public key from [X9.62] MUST NOT be used.

As specified in SEC1:2009:

- The elliptic curve public key (a value of type ECPoint that is an OCTET STRING) is mapped to a subjectPublicKey (a value of type BIT STRING) as follows: the most significant bit of the OCTET STRING value becomes the most significant bit of the BIT STRING value, and so on; the least significant bit of the OCTET STRING becomes the least significant bit of the BIT STRING. Conversion routines are found in Sections 2.3.1 and 2.3.2 of SEC1:2009;
- The first octet of the OCTET STRING indicates whether the key is compressed or uncompressed. The uncompressed form is indicated by 0x04 and the compressed form is indicated by either 0x02 or 0x03 (see 2.3.3 in SEC1:2009). The public key MUST be rejected if any other value is included in the first octet.

#### 9.2.6.4.3.6 Subject Unique ID

Subject unique IDs may be optionally used in end device certificates other than server certificates. The use of this field is left to project specific companion specifications.

#### 9.2.6.4.4 Certificate extensions

##### 9.2.6.4.4.1 Overview

The extensions defined for X.509 v3 certificates provide methods for associating additional attributes with users or public keys and for managing relationships between CAs. Each extension in a certificate is designated as either critical (TRUE) or non-critical (FALSE).

The extension fields have to be completed according to the type of Certificate used, as specified in Table 36.

**Table 36 – X.509 v3 Certificate extensions**

	Attributes	RFC 5280 reference	Clause	CAs		End entities		
				C (Root)	C (Sub-CA)	C (TLS)	C (Key Agree)	C (Data Sign)
1	AuthorityKeyIdentifier	4.2.1.1	9.2.6.4.4.2	o	m	m	m	m
2	SubjectkeyIdentifier	4.2.1.2	9.2.6.4.4.3	m	m	o	o	o
3	KeyUsage	4.2.1.3	9.2.6.4.4.4	m	m	m	m	m
4	CertificatePolicies	4.2.1.4	9.2.6.4.4.5	o	m	m	o	o
5	SubjectAltNames	4.2.1.6	9.2.6.4.4.6	o	o	o	o	o
6	IssuerAltNames	4.2.1.7	9.2.6.4.4.7	o	o	x	x	x
7	BasicConstraints	4.2.1.9	9.2.6.4.4.8	m	m	x	x	x
8	ExtendedKeyUsage	4.2.1.12	9.2.6.4.4.9	x	x	m	x	x
9	cRLDistributionPoints	4.2.1.13	9.2.6.4.4.10	o	o	x	x	x

C(Root): Certificate of the Root CA

C(Sub-CA): Certificate of a Sub-CA

C(TLS): Certificate for Transport Layer Security

C(KeyAgree): Certificate an ECDH capable key establishment key

C(DataSign): Certificate of an ECDSA capable signing key

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	239/614
-----------------------	------------	-----------------------	---------

#### 9.2.6.4.4.2 Authority Key Identifier

- Extension-ID (OID): 2.5.29.35;
- Critical: FALSE;
- Description: the AuthorityKeyIdentifier extension provides a means of identifying the public key corresponding to the private key used to sign a certificate;
- Value: the AuthorityKeyIdentifier extension MUST include the `keyIdentifier` field.

For the method of calculating the `keyIdentifier` see 9.2.6.4.4.3.

NOTE The choice of the method is left to project specific companion specifications.

#### 9.2.6.4.4.3 SubjectKeyIdentifier

- Extension-ID (OID): 2.5.29.14;
- Critical: FALSE;
- Description: the SubjectKeyIdentifier extension provides a means of identifying certificates that contain a particular public key;
- Value: the SubjectKeyIdentifier extension MUST include the `keyIdentifier` field.

The value of the `keyIdentifier` field needs to be computed either:

- with the method 1 defined in RFC 5280, 4.2.1.2 i.e. the `keyIdentifier` is composed of the 160-bit SHA-1 hash of the value of the BIT STRING `SubjectPublicKey` (excluding the tag, length, and number of unused bits); or
- with the method 2 defined in RFC 5280, 4.2.1.2 i.e. the `keyIdentifier` is composed of a four-bit type field with the value 0100 followed by the least significant 60 bits of the SHA-1 hash of the value of the BIT STRING `SubjectPublicKey` (excluding the tag, length, and number of unused bits).

#### 9.2.6.4.4 KeyUsage

- Extension-ID (OID): 2.5.29.15;
- Critical: TRUE;
- Description: the KeyUsage extension defines the purpose of the key contained in the certificate;
- Value: The bits that shall be set are shown in Table 37.

**Table 37 – Key Usage extensions**

Certificate	<i>C (Root)</i>	<i>C (Sub-CA)</i>	<i>C (TLS)</i>	<i>C (KeyAgree)</i>	<i>C (DataSign)</i>
Bits to be set	keyCertSign, cRLSign	keyCertSign, cRLSign	digitalSignature keyAgreement	keyAgreement	digitalSignature

For details, see RFC 5280, 4.2.1.3 and NSA3.

#### 9.2.6.4.4.5 CertificatePolicies

- Extension-ID (OID): 2.5.29.32;
- Critical: FALSE;
- Description: the certificate policies extension contains a sequence of one or more policy information terms, each of which consists of an object identifier (OID) and optional qualifier. For details, see RFC 5280, 4.2.1.4;
- Value: contains the OID for the applicable certificate policy.

#### 9.2.6.4.4.6 SubjectAltNames

- Extension-ID (OID): 2.5.29.17;
- Critical: TRUE if the “subject” field of the certificate is empty (an empty sequence), else FALSE.

Description: this extension allows identities to be bound to the subject of the certificate. These identities may be included in addition to or in place of the identity in the subject field of the certificate.

If the subject name is an empty sequence, then the `subjectAltName` extension MUST be added in the End Entity Signature and Key Establishment Certificates and MUST be marked as critical. The `subjectAltName` extension is OPTIONAL otherwise and if included, MUST be marked as critical.

The `SubjectAltName` extension when used shall contain a single `GeneralName` of type `OtherName` that is further sub-typed as a `HardwareModuleName` (`id-on-HardwareModule`) as defined in IETF RFC 4108. The `hwSerialNum` field shall be set to the system title.

- Value: See Table 38.

**Table 38 – Subject Alternative Name values**

Certificate	<i>C (Root)</i>	<i>C (Sub-CA)</i>	<i>C (TLS)</i>	<i>C (KeyAgree)</i>	<i>C (DataSign)</i>
rfc822Name	<E-Mail Address>	<E-Mail Address>	-	-	-
URI	<Web site>	<Web site>	-	-	-
otherName	-	-	-	<otherName>	<otherName>
NOTE Values within the less-than – greater-than signs “< >” are to be assigned by the PKI or the CA as applicable.					

#### 9.2.6.4.4.7 IssuerAltName

- Extension-ID (OID): 2.5.29.18;
- Critical: FALSE;
- Description: this extension is used to associate Internet style identities with the certificate issuer. For details see RFC 5280, 4.2.1.7;
- Value: See Table 39.

**Table 39 – Issuer Alternative Name values**

Certificate	<i>C (Root)</i>	<i>C (Sub-CA)</i>	<i>C (TLS)</i>	<i>C (KeyAgree)</i>	<i>C (dataSign)</i>
rfc822Name	<E-Mail Address>	<E-Mail Address>	-	-	-
URI	<Web site>	<Web site>	-	-	-
NOTE Values within the less-than – greater-than signs “< >” are to be assigned by the PKI or the CA as applicable.					

#### 9.2.6.4.4.8 Basic constraints

- Extension-ID (OID): 2.5.29.19;
- Critical: TRUE;
- Description: the basic constraints extension identifies whether the subject of the certificate is a CA and the maximum depth of valid certification paths that include this certificate. See RFC 5280, 4.2.1.9;
- Value: See Table 40.

**Table 40 – Basic constraints extension values**

Certificate	<i>C (Root)</i>	<i>C (Sub-CA)</i>	<i>C (TLS)</i>	<i>C (KeyAgree)</i>	<i>C (dataSign)</i>
ca	TRUE	TRUE	-	-	-
pathLenConstraint	See Note 1	See Note 1	-	-	-
NOTE 1 The value of the – optional – pathLenConstraint depends on the structure of the PKI.					

#### 9.2.6.4.4.9 Extended Key Usage

- Extension-ID (OID): 2.5.29.37;
- Critical: FALSE;
- Description: Indicates that a certificate can be used as an TLS server certificate;
  - TLS server authentication OID: 1.3.6.1.5.5.7.3.1;
  - TLS client authentication OID: 1.3.6.1.5.5.7.3.2.

#### 9.2.6.4.4.10 cRLDistributionPoints

- Extension-ID (OID): 2.5.29.31;
- Critical: FALSE;
- Description: The CRL distribution points extension identifies how CRL information is obtained;
- This extension is not used in DLMS server certificates.

#### 9.2.6.4.4.11 Other extensions

All other extensions not described in this profile should be considered OPTIONAL; their inclusion or exclusion and their values will depend upon the particular application or PKI profile.

### 9.2.6.5 Suite B end entity certificate types to be supported by DLMS servers

Every DLMS server must use X.509 v3 format and contain either:

- a P-256 or P-384 ECDSA-capable signing key; or
- a P-256 or P-384 ECDH-capable key agreement key.

Every certificate must be signed using ECDSA. The signing CA's key must be P-256 or P-384 if the certificate contains a key on P-256. The signing CA's key must be P-384 if the certificate contains a key on P-384.

Depending on the security policy, the following X.509 v3 certificates shall be handled by DLMS/COSEM end entities, see Table 41:

**Table 41 – Certificates handled by DLMS/COSEM end entities**

Security suite 1	Security suite 2	Role
Root Certification Authority (Root-CA) Self-Signed Certificate using P-256 signed with P-256	Root Certification Authority (Root-CA) Self-Signed Certificate using P-384 signed with P-384	Trust anchor; there may be more than one.
Subordinate CA Certificate (Sub-CA) using P-256 signed with P-256	Subordinate CA Certificate (Sub-CA) using P-384 signed with P-384	Certificate of an issuing CA. Subordinate CAs may be also used as trust anchors.
Subordinate CA Certificate (Sub-CA) using P-256 signed with P-384	–	
End-Entity Signature Certificate using P-256 signed with P-256	End-Entity Signature Certificate using P-384 signed with P-384	Public key for ECDSA signature generation and verification
End-Entity Signature Certificate using P-256 signed with P-384	–	
End-Entity Key Establishment Certificate using P-256 signed with P-256	End-Entity Key Establishment Certificate using P-384 signed with P-384	Used with the One-Pass Diffie-Hellman C(1e, 1s) scheme or with the Static Unified Model C(0e, 2s, ECC CDH) scheme
End-Entity Key Establishment Certificate using P-256 signed with P-384	–	
End-Entity TLS Certificate using P-256 signed with P-256	End-Entity TLS Certificate using P-384 signed with P-384	
End-Entity TLS Certificate using P-256 signed with P-384	–	

An example Certificate is given in Annex B.

### 9.2.6.6 Management of certificates

#### 9.2.6.6.1 Overview

This subclause 9.2.6.6 applies only to the management of public key certificates in DLMS servers, including:

- provisioning the server with trust anchors, see 9.2.6.6.2;
- provisioning the server with further CA certificates, see 9.2.6.6.3;
- security personalisation of the server, see 9.2.6.6.4;
- provisioning servers with certificates of clients and third parties, see 9.2.6.6.5;
- provisioning clients and third parties with certificates of servers, see 9.2.6.6.6;
- removing certificates, see 9.2.6.6.7.

**NOTE** Management of public key certificates in DLMS clients and in third party systems is out of the Scope of this Technical Report.

#### 9.2.6.6.2 Provisioning servers with trust anchors

Before starting steady state operations, servers shall be provisioned with trust anchors that will be used to validate the certificates. Trust anchors may be Root-CA (i.e. self-signed) certificates, Sub-CA certificates or directly trusted CA keys. A server may be provisioned with more than one trust anchor.

Trust anchors shall be placed in the server out of band (OOB).

Trust anchor certificates are stored together with other certificates.

They can be exported, but they cannot be imported or removed.

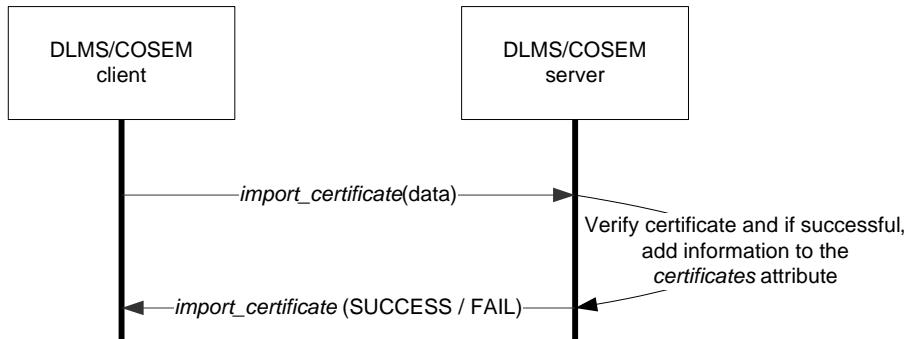
Directly trusted CA keys cannot be exported.

#### 9.2.6.6.3 Provisioning the server with further CA certificates

The server may be provisioned with further CA certificates that will be used to verify digital signatures on end device certificates.

For this purpose the *import\_certificate* method of the “Security setup” object is available. The process is shown in Figure 101.

Precondition: The DLMS/COSEM client verified the CA certificate.  
The server has been provisioned with trust anchor(s).



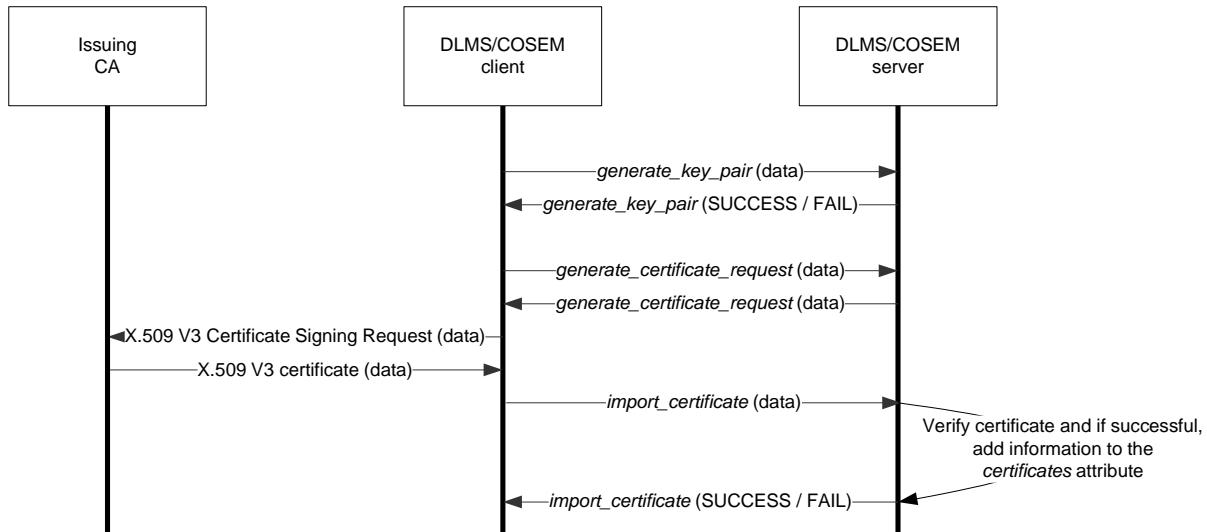
**NOTE** When a third party is responsible for managing CA certificates, then the *import\_certificate* method may be invoked by that third party via the client acting as a broker.

**Figure 101 – MSC for provisioning the server with CA certificates**

#### 9.2.6.6.4 Security personalisation of the server

Security personalisation means the provision of the server with asymmetric key pairs and the corresponding public key certificates. This can take place either:

- using the security primitives provided by the manufacturer to inject the private key and the public key certificates. The private keys have to be securely stored in the server and shall never be exposed; or
- using the appropriate methods of a “Security setup” object. This process can be used during the manufacturing process and whenever a new key pair and the related public key certificate need to be generated. This process is shown in Figure 102 and described below.



NOTE The security personalisation may be carried out by a third party instead of the client. In that case, the methods of the “Security setup” object may be invoked by that third party via the client acting as a broker.

**Figure 102 – MSC for security personalisation of the server**

- Step 1: the client invokes the `generate_key_pair` method. The method invocation parameters specify the key pair to be generated: digital signature, key agreement or TLS key pair;

NOTE 1 The new key pair can be used in transactions once its certificate will have been imported and successfully verified.

- Step 2: the client invokes the `generate_certificate_request` method. The method invocation parameters identify the key pair for which the Certificate Signing Request (CSR) will be generated. The return parameters include the CSR, signed by the private key of the newly generated key pair;
- Step 3: The client sends the CSR to the CA. This message shall encapsulate the return parameters resulting from the invocation of the `generate_certificate_request` method. The CA – provided that the necessary conditions are met – issues the certificate and sends it to the client;

NOTE 2 The format of the messages between the client and the issuing CA is out of the Scope of this Technical Report.

- Step 4: The client invokes the `import_certificate` method. The method invocation parameters contain the certificate. The server verifies the certificate and if successful adds information on the certificate to the `certificates` attribute. If the verification fails, the certificate shall be discarded.

There may be only one key pair and certificate present for the same purpose (digital signature, key agreement, TLS). Therefore when the new certificate is successfully imported the old certificate is removed. From this point, the new key pair can be used for transactions.

For the details of method invocation and return parameters, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7.

Parties using the server's certificates can obtain these either:

- out of band;
- using the `export_certificate` method of the “Security setup” objects, see 9.2.6.6.6; or
- as part of the AARE (during HLS authentication), see 9.2.7.4.

### 9.2.6.6.5 Provisioning servers with certificates of clients and third parties

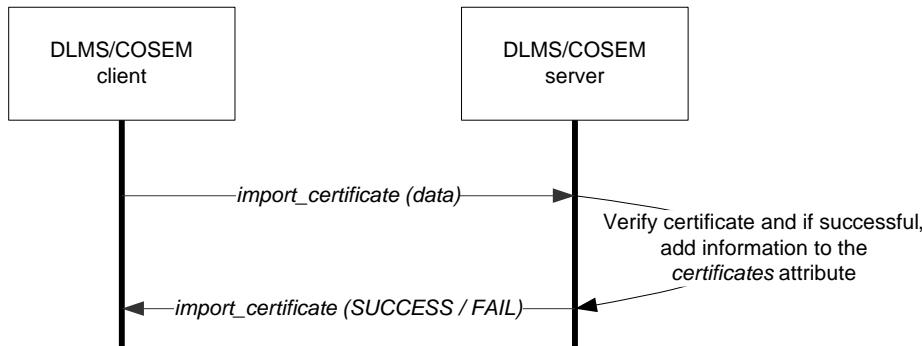
To verify digital signatures, to perform key agreement using a scheme that uses static key agreement keys, or to establish a TLS connection, the server needs to have the appropriate public key certificates of the other party.

If, at the time of manufacturing, the client and/or third parties are already known, their public keys certificates may be injected into the server by the manufacturer.

**NOTE** Distribution of public key certificates to the manufacturer is out of Scope of this Technical Report.

Otherwise, the servers can be provisioned with certificates of clients and third parties using the *import\_certificate* method of the “Security setup” object. The method invocation parameter is the certificate to be placed in the server. For the details of method invocation and return parameters, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7. The process is shown in Figure 103.

Precondition: The DLMS/COSEM client verified the certificate.  
The server has been provisioned with the chain of CA certificates.



**NOTE** The *import\_certificate (data)* method may be also invoked by a third party, using the client as a broker.

**Figure 103 – Provisioning the server with the certificate of the client**

In the case of HLS authentication mechanism using ECDSA, the public key certificate of the clients' digital signature key can be carried by the calling-AE-qualifier field of the AARQ. See 9.2.7.4.

### 9.2.6.6 Provisioning clients and third parties with certificates of servers

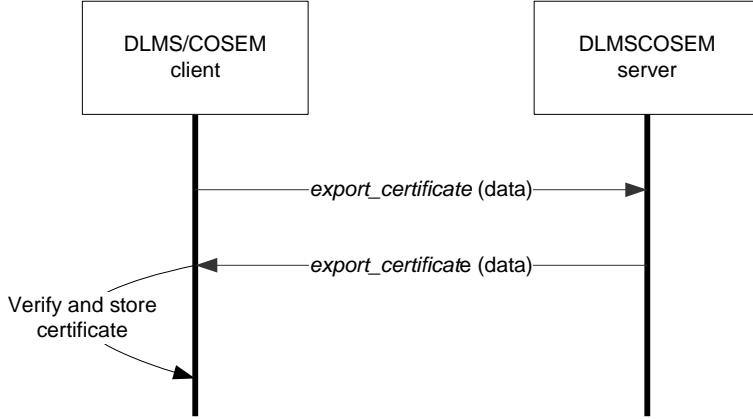
To verify digital signatures applied by the server, to perform key agreement that involves a static key agreement key, or to establish a TLS connection the client or third party needs to have the appropriate public key certificate of the server.

The certificate may be delivered with the server and inserted in clients / third parties OOB.

Alternatively, the client or third party can request the certificate from the server using the *export\_certificate* method of the “Security setup” object. The method invocation parameter identifies the certificate requested.

**NOTE** In the case of HLS authentication using ECDSA – this is authentication\_mechanism 7 – the public key certificate of the server for digital signature is transported in the AARE.

The return parameters – in the case of success – include the certificate. For the details of method invocation and return parameters, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7. The process is shown in Figure 104.



NOTE The *export\_certificate (data)* method may be also invoked by a third party, using the client as a broker.

**Figure 104 – Provisioning the client / third party with a certificate of the server**

#### 9.2.6.6.7 Certificate removal from the server

It is sometimes necessary to remove a public key certificate stored by the server.

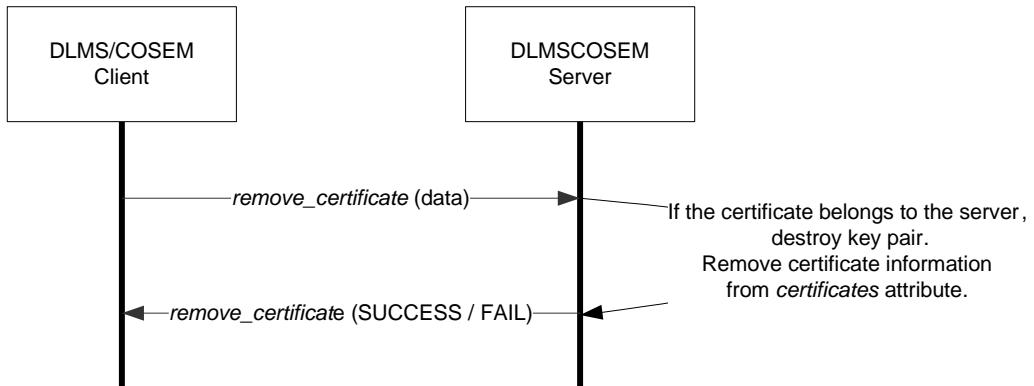
NOTE 1 This may relate to certificates that belong to the server or certificates that belong to a client or third party.

NOTE 2 The conditions when removal of a public key certificate is necessary are out of the Scope of this Technical Report.

When a certificate that belongs to the server is removed, the private key associated with the public key shall be destroyed.

The information on the certificate removed shall be also removed from the *certificates* attribute of the "Security setup" object.

The key pair the public key certificate of which has been removed cannot be used any more for transactions. The process is shown in Figure 105.



NOTE The *remove\_certificate (data)* method may be also invoked by a third party, using the client as a broker.

**Figure 105 – Remove certificate from the server**

#### 9.2.7 Applying cryptographic protection

##### 9.2.7.1 Overview

The cryptographic algorithms specified in 9.2.3 can be applied:

- to protect the xDLMS APDUs see 9.2.7.2;
- to process the challenges during HLS authentication, see 9.2.7.4;
- to protect COSEM data, see 9.2.7.5.

### 9.2.7.2 Protecting xDLMS APDUs

#### 9.2.7.2.1 Overview

This subclause 9.2.7.2 specifies how the cryptographic algorithms specified in 9.2.3.3 and 9.2.3.4 can be used to protect xDLMS APDUs:

- 9.2.7.2.2 specifies the possible values of security policy and access rights;
- 9.2.7.2.3 presents the types of ciphered APDUs;
- 9.2.7.2.4 specifies the use of the AES-GCM algorithm for authentication and encryption;
- 9.2.7.2.5 specifies the use of the ECDSA algorithm for digital signature.

When a COSEM object attribute or method related xDLMS service primitive is invoked by the AP, the service parameters include the Security\_Options parameter. This parameter informs the AL on the kind of ciphered APDU to be used, on the protection to be applied, and includes the necessary security material. The AL builds first the APDU corresponding to the service primitive then it builds the ciphered APDU.

When the AL receives a ciphered APDU from a remote partner, it deciphers it and restores the original, unciphered APDU. When this is successfully done, it invokes the appropriate service primitive. The additional service parameters include the Security\_Status and the Protection\_Element parameters that inform the AP about the kind of ciphered APDU used, on the protection that has been verified and removed, and may include security material. See also 9.3.5.

#### 9.2.7.2.2 Security policy and access rights values

In the case of “Security setup” version 1 – see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.7 – the enum type shall be interpreted as an unsigned 8 type; the meaning of each bit is as shown in Table 42.

**Table 42 – Security policy values (“Security setup” version 1)**

Bit	Security policy
0	unused, shall be set to 0
1	unused, shall be set to 0
2	authenticated request
3	encrypted request
4	digitally signed request
5	authenticated response
6	encrypted response
7	digitally signed response

NOTE In the case of “Security policy” version 0 the possible security policy values are specified in DLMS UA 1000-1 Part 2 Ed.15:2021, 5.4.8. For both “Security policy” version 0 and 1 the value (0) means that no cryptographic protection is required.

Access rights are held by the *object\_list* attribute of “Association LN” or the *access\_rights\_list* of “Association SN” objects. The *access\_mode* element of the *access\_rights* determines the access kind and stipulates the cryptographic protection. It is an *enum* data type.

In the case of “Association LN” version 3 and “Association SN” version 4 – see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.4. and 4.4.3 – the *enum* value is interpreted as an *unsigned8* and the meaning of each bit is as shown in Table 43.

For older versions, see their specification in DLMS UA 1000-1, the “Blue Book”.

**Table 43 – Access rights values (“Association LN” ver 3 “Association SN” ver 4)**

Bit	Attribute access	Method access
0	read-access	access

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	247/614
-----------------------	------------	-----------------------	---------

Bit	Attribute access	Method access
1	write-access	not-used
2	authenticated request	authenticated request
3	encrypted request	encrypted request
4	digitally signed request	digitally signed request
5	authenticated response	authenticated response
6	encrypted response	encrypted response
7	digitally signed response	digitally signed response
Examples	enum (3): read-write enum (6) write access with authenticated request enum (255): read-write access with authenticated, encrypted and digitally signed request and response	enum (1): access enum (2): not used enum (5): access with authenticated request enum (253): access with authenticated, encrypted and digitally signed request and response

Access rights to COSEM object attributes and/or methods may require authenticated, encrypted and / or signed xDLMS APDUs. For this reason, APDUs with more protection than required by the security policy are always allowed. APDUs with less protection than required by the security policy and the access rights shall be rejected.

More protection in this context means that more kinds of protection are applied on the xDLMS APDU than what is requested by the security policy: for example, security policy requires that all APDUs are authenticated but access rights require that the APDU is encrypted and authenticated i.e. a higher protection.

#### 9.2.7.2.3 Ciphered xDLMS APDUs

The different kind of ciphered xDLMS APDUs are shown in Table 44. See also 9.3.5.

Ciphered xDLMS APDUs can be used in a ciphered application context only. On the other hand, in a ciphered application context, both ciphered and unciphered APDUs may be used.

**Table 44 – Ciphered xDLMS APDUs**

APDU type	Parties	Type of ciphersing	Security services	Key used	Com-pression
Service-specific glo-ciphering or ded-ciphering				Block cipher key: - Dedicated key <sup>1</sup>	—
general-glo-ciphering general-ded-ciphering	Client – Server	Symmetric key	Authentication Encryption	- Global unicast / broadcast key established <sup>2</sup> outside the exchange <sup>3</sup> , identified by the SC byte  Authentication key: global, established <sup>2</sup> outside the exchange <sup>3</sup>	Yes <sup>5</sup>
general-ciphering	Third party or Client – Server	Symmetric key	Authentication Encryption	Block cipher key: - Global unicast / broadcast, established <sup>2</sup> outside the exchange <sup>3</sup> , identified as part of the exchange, or - Established <sup>2</sup> as part of the exchange <sup>4</sup>  Authentication key: global, established <sup>2</sup> outside the exchange <sup>3</sup>	Yes <sup>5</sup>
general-signing		Asymmetric key	Digital signature	Signing key	No

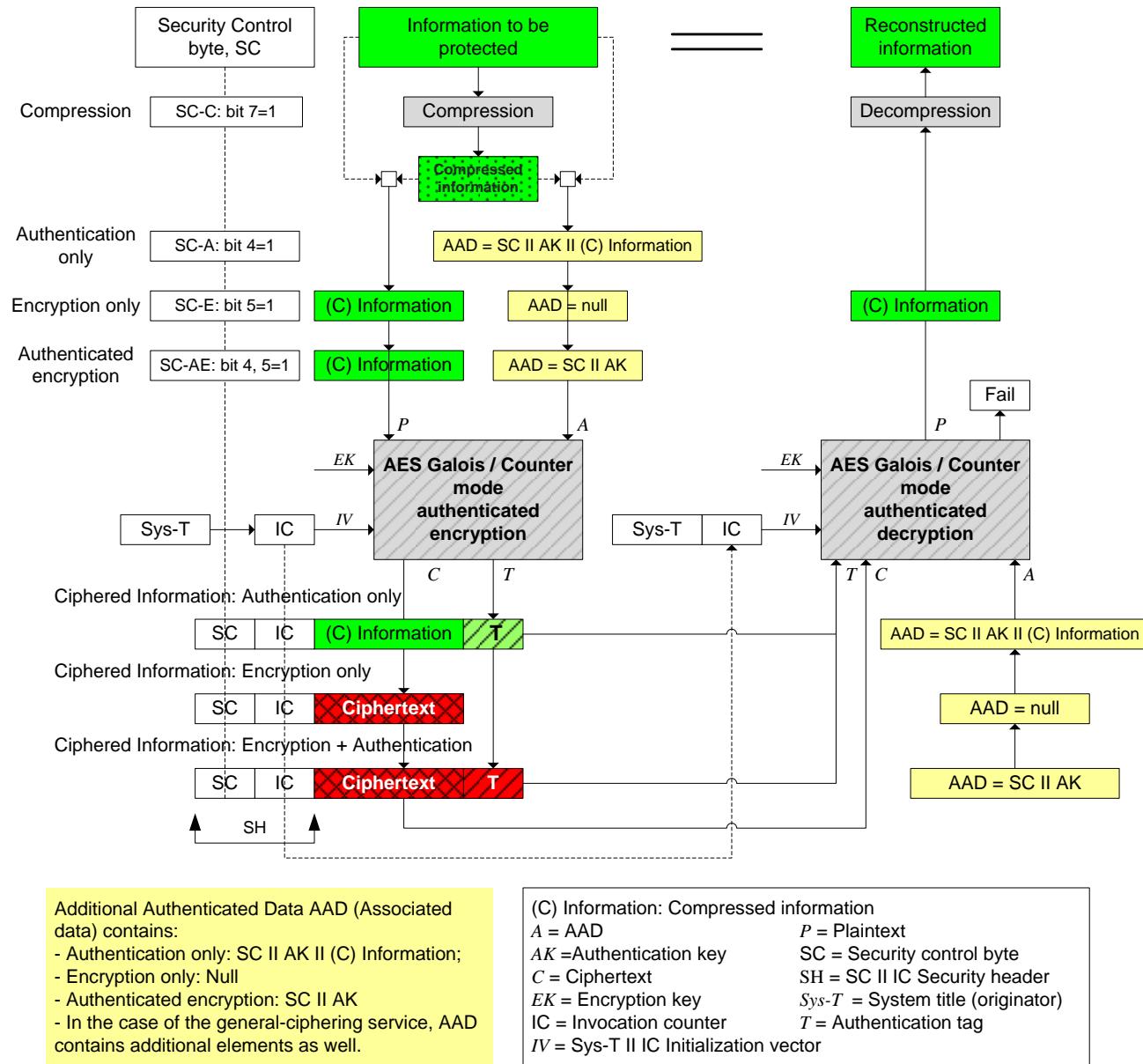
<sup>1)</sup> Transported by the AARQ;  
<sup>2)</sup> Key establishment may be key wrapping or key agreement;  
<sup>3)</sup> In the server, these keys are held by the Security setup objects;  
<sup>4)</sup> Key data is transported in the protected APDU;  
<sup>5)</sup> The use of compression is controlled by the Security Control byte.

### 9.2.7.2.4 Encryption, authentication and compression

#### 9.2.7.2.4.1 Overview

Encryption and authentication to protect information using the AES-GCM algorithm is shown in Figure 106. See also 9.2.3.3.7. This algorithm can be combined with compression.

In the case of message protection, the information to be protected is an xDLMS APDU. In the case of COSEM data protection, the information to be protected is COSEM data, i.e. COSEM attribute value(s) or method invocation / return parameter(s).



NOTE In the case of general-ciphering, AAD also includes additional fields, see Table 46.

**Figure 106 – Cryptographic protection of information using AES-GCM**

The security material required is specified in 9.2.7.2.4.2 – 9.2.7.2.4.5.

#### 9.2.7.2.4.2 The security header

The security header SH includes the security control byte concatenated with the invocation counter: SH = SC II IC. The security control byte is shown in Table 45 where:

- Bit 3...0: Security\_Suite\_Id, see 9.2.3.7;
- Bit 4: "A" subfield: indicates that authentication is applied;
- Bit 5: "E" subfield: indicates that encryption is applied;
- Bit 6: Key\_Set subfield:
  - 0 = Unicast,
  - 1 = Broadcast;
- Bit 7: Indicates the use of compression.

**Table 45 – Security control byte**

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3..0
Compression	Key_Set	E	A	Security_Suite_Id
The Key_Set bit is not relevant and shall be set to 0 when the service-specific dedicated ciphering, the general-ded-ciphering or the general-ciphering APDUs are used.				

#### 9.2.7.2.4.3 Plaintext and Additional Authenticated Data

The plaintext denoted  $P$  and the Additional Authenticated Data denoted  $A$  depend on the kind of protection. They are shown in Table 46, where  $SC$  is the security control byte,  $AK$  is the authentication key and  $I$  is the information, i.e. the xDLMS APDU or the COSEM data to be protected.

**Table 46 – Plaintext and Additional Authenticated Data**

Security control, $SC$		Protection	$P$	$A$ , Additional Authenticated Data	
E field	A field			Service-specific glo-ciphering Service-specific ded-ciphering general-glo-ciphering general-ded-ciphering	general-ciphering
0	0	None	–	–	–
0	1	Authenticated only	–	$SC \text{ II } AK \text{ II } (C) I$	$SC \text{ II } AK \text{ II }$ transaction-id II <sup>1</sup> originator-system-title II <sup>1</sup> recipient-system-title II <sup>1</sup> date-time II <sup>1</sup> other-information II <sup>1</sup> (C) I
1	0	Encrypted only	(C) I	–	–
1	1	Encrypted and authenticated	(C) I	$SC \text{ II } AK$	$SC \text{ II } AK \text{ II }$ transaction-id II <sup>1</sup> originator-system-title II <sup>1</sup> recipient-system-title II <sup>1</sup> date-time II <sup>1</sup> other-information

1) The fields transaction-id ...other-information are A-XDR encoded OCTET STRINGS. The length and the value of each field is included in the AAD.

#### 9.2.7.2.4.4 Encryption key and authentication key

These keys used by AES-GCM are specified in 9.2.3.3.7.4 and 9.2.3.3.7.5 respectively. The various keys used in DLMS/COSEM and their establishment are specified in 9.2.5.

250/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

#### 9.2.7.2.4.5 Initialization vector

See 9.2.3.3.7.3.

#### 9.2.7.2.4.6 Service-specific ciphering xDLMS APDUs

For certain xDLMS APDUs – see 9.1.4.4.6 and 9.5 – a ciphered variant using the global key and one using the dedicated key is available. These ciphered APDUs can be used between a client and a server. The structure of the service-specific ciphering APDUs is shown in Figure 107. See also Table 46 and Table 47.

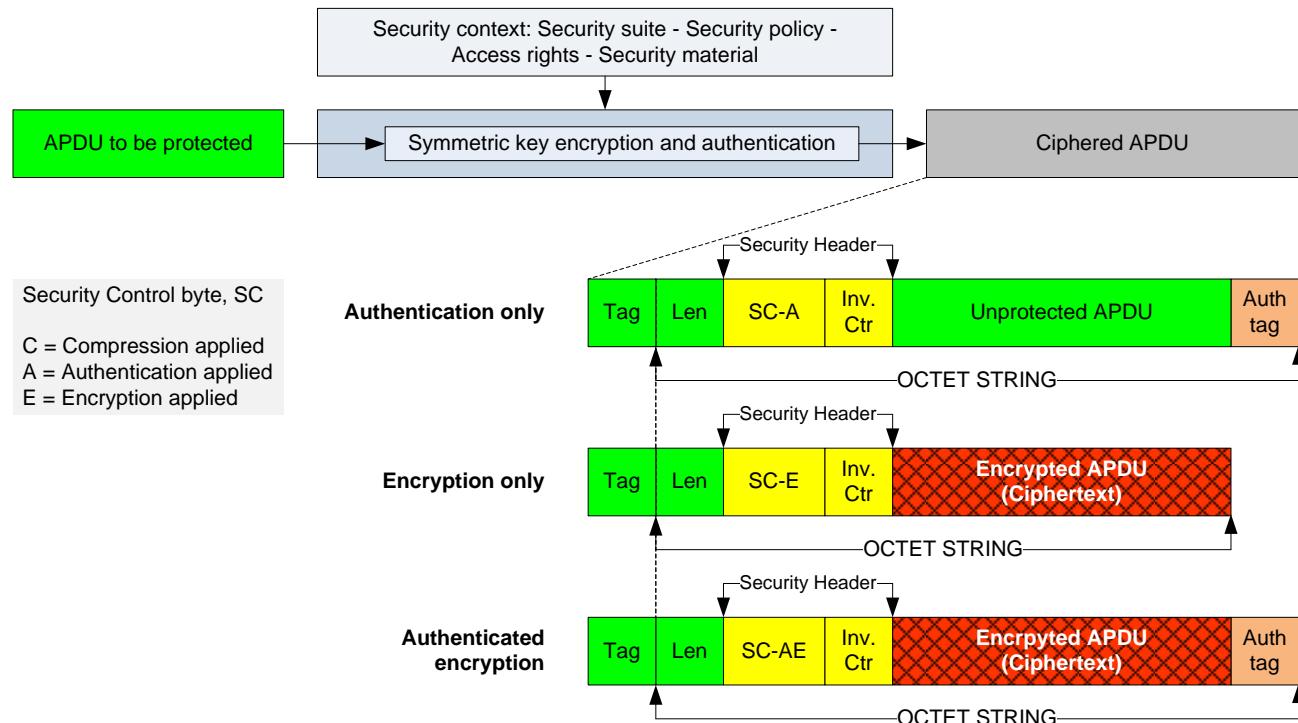
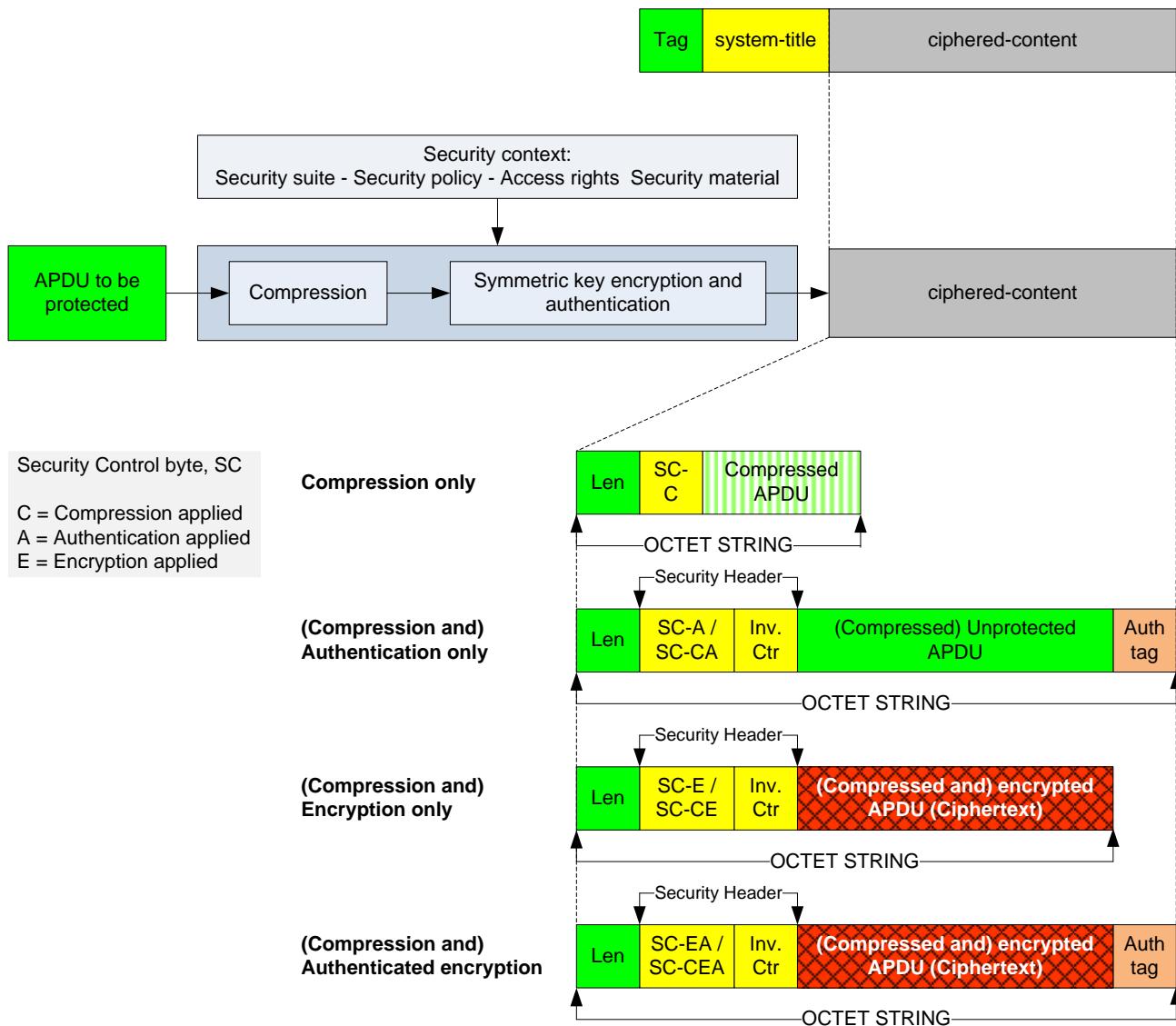


Figure 107 – Structure of service-specific global / dedicated ciphering xDLMS APDUs

#### 9.2.7.2.4.7 The general-glo-ciphering and general-ded-ciphering xDLMS APDUs

These APDUs can be used to cipher other xDLMS APDUs using the global key or the dedicated key. They can be used between a client and a server. Their structure is shown in Figure 108. See also Table 46 and Table 47.

**Figure 108 – Structure of general-glo-ciphering and general-ded-ciphering xDLMS APDUs****9.2.7.2.4.8 The general-ciphering APDU**

The general-ciphering APDU can be used between a client and a server or between a third party and the server. These APDUs carry also the necessary information on the key to be used. The structure of the general-ciphering APDU is shown in Figure 109. See also Table 46 and Table 47.

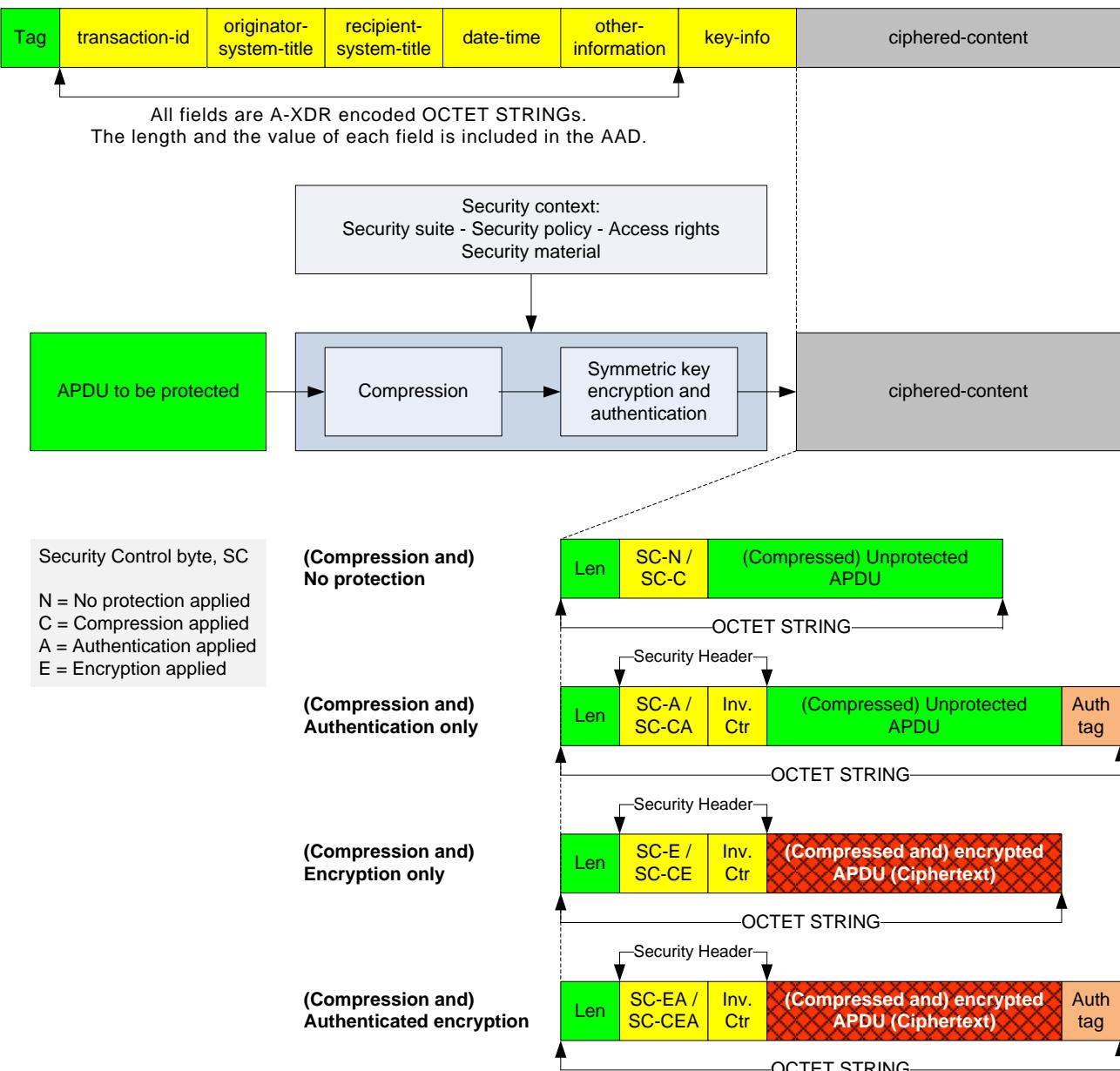


Figure 109 – Structure of general-ciphering xDLMS APDUs

#### 9.2.7.2.4.9 Use of the fields of the ciphering xDLMS APDUs

The use of the fields of the ciphering xDLMS APDUs is summarized in Table 47.

Table 47 – Use of the fields of the ciphering xDLMS APDUs

APDU field	Service-specific global / dedicated ciphering	general-glo-/ general-ded-ciphering	general-ciphering	Meaning
tag	Service specific	[219] [220]	[221]	The tag of the ciphering APDU; see 9.5
system-title	–	+	–	See 9.3.5.

APDU field	Service-specific global / dedicated ciphering	general-glo-/ general-ded- ciphering	general-ciphering	Meaning
transaction-id	–	–	+	
originator-system-title	–	–	+	
recipient-system-title	–	–	+	
date-time	–	–	+	
other-information	–	–	+	
key-info	–	–	+	
security control byte	+	+	+	Provides information on the protection applied, the key-set and the security suite used. See Table 45. <sup>1)</sup>
Invocation counter	+	+	+	The invocation field of the initialization vector. It is an integer counter which increments upon each invocation of the authenticated encryption function using the same key. When a new key is established the related invocation counter shall be reset to 0.
unprotected APDU	+	+	+	The unprotected APDU (same as the APDU to be protected).
encrypted APDU	+	+	+	The encrypted APDU i.e. the ciphertext.
authentication tag	+	+	+	Calculated by the AES-GCM algorithm, see 9.2.3.3.7.
<sup>1)</sup> In the case of the general-ciphering APDU, the key-set bit of the security control byte is not relevant and shall be set to zero.				

#### 9.2.7.2.4.10 Encoding example: global-get-request xDLMS APDU

Table 48 shows an encoding example of a service-specific global ciphering xDLMS APDU: glo-get-request.

**Table 48 – Example: glo-get-request xDLMS APDU**

	X	Contents	LEN (X) bytes	Len (X) bits
<b>Security material</b>				
Security suite		GCM-AES-128		
System Title	Sys-T	<b>4D4D4D0000BC614E</b> (here, the five last octets contain the manufacturing number in hexa)	8	64
Invocation counter	IC	01234567	4	32
Initialization Vector	IV	<i>Sys-T    IC</i> <b>4D4D4D0000BC614E01234567</b>	12	96
Block cipher key (global)	EK	000102030405060708090A0B0C0D0E0F	16	128
Authentication Key	AK	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	16	128

	<i>X</i>	<b>Contents</b>			<i>LEN</i> ( <i>X</i> ) <i>bytes</i>	<i>Len</i> ( <i>X</i> ) <i>bits</i>
<b>Security applied</b>		<b>Authentication</b>	<b>Encryption</b>	<b>Authenticated encryption</b>		
Security control byte (with unicast key)	<i>SC</i>	<i>SC-A</i>	<i>SC-E</i>	<i>SC-AE</i>	1	8
		10	20	30		
Security header	<i>SH</i>	<i>SH = SC-A    IC</i>	<i>SH = SC-E    IC</i>	<i>SH = SC-AE    IC</i>		
		1001234567	2001234567	3001234567	5	40
<b>Inputs</b>		<b>Authentication</b>	<b>Encryption</b>	<b>Authenticated encryption</b>		
xDLMS APDU to be protected	<i>APDU</i>	C0010000080000010000FF0200 (Get-request, attribute 2 of the Clock object)			13	104
Plaintext	<i>P</i>	Null	C0010000080000010000FF0200	C0010000080000010000FF0200	13	104
Associated data	<i>A</i>	<i>SC    AK    APDU</i>	-	<i>SC    AK</i>		
Associated Data – Authentication	<i>A-A</i>	10D0D1D2D3D4D5D6 D7D8D9DADBCDDDE DFC0010000080000010000FF0200	-	-	30	240
Associated Data – Encryption	<i>A-E</i>	-	-	-	0	0
Associated Data – Authenticated encryption	<i>A-AE</i>	-	-	30D0D1D2D3D4D5D6 D7D8D9DADBCDDDE DF	17	136
<b>Outputs</b>		<b>Authentication</b>	<b>Encryption</b>	<b>Authenticated encryption</b>		
Ciphertext	<i>C</i>	NULL	411312FF935A4756 6827C467BC	411312FF935A4756 6827C467BC	13	104
Authentication tag	<i>T</i>	06725D910F9221D2 63877516	-	7D825C3BE4A77C3F CC056B6B	12	96
The complete Ciphered APDU		<i>TAG    LEN    SH    APDU    T</i>	<i>TAG    LEN    SH    C</i>	<i>TAG    LEN    SH    C    T</i>	-	-
Authenticated APDU		C81E1001234567C0 0100000800000100 00FF020006725D91 0F9221D263877516	-	-	32	256
Encrypted APDU		-	C812200123456741 1312FF935A475668 27C467BC	-	20	160
Authenticated and encrypted APDU		-	-	C81E300123456741 1312FF935A475668 27C467BC7D825C3B E4A77C3FCC056B6B	32	256
NOTE In this example the value of the invocation counter is 01234567. In the real life, the value shall be incremented with each invocation of the AES-GCM algorithm.						

Table 49 shows an example where the ACCESS.request and ACCESS.response APDUs shown in Table 148 are protected using authenticated encryption. The general-ciphering APDU specified in 9.2.7.2.4.8 is used. The encryption key is agreed on using the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) key agreement scheme, see 9.2.3.4.6.3. The authentication key is the same as in Table 48.

**Table 49 – ACCESS service with general-ciphering, One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) key agreement scheme**

Message Elements	Contents	LEN (Bytes)
<b>General-Ciphering</b>	DD	1
<b>transaction-id</b>		0
<b>length</b>	08	1
<b>value</b>	0102030405060708	8
<b>originator-system-title</b>		0
<b>length</b>	08	1
<b>value</b>	4D4D4D0000BC614E	8
<b>recipient-system-title</b>		0
<b>length</b>	08	1
<b>value</b>	4D4D4D0000000001	8
<b>date-time</b>		0
<b>length</b>	00	1
<b>value</b>		0
<b>other-information</b>		0
<b>length</b>	00	1
<b>value</b>		0
<b>key-info OPTIONAL</b>	01	1
<b>agreed-key CHOICE</b>	02	1
<b>key-parameters</b>		0
<b>length</b>	01	1
<b>value</b>	01	1
<b>key-ciphered-data</b>		0
<b>length</b>	8180	2
<b>value</b>	C323C2BD45711DE4688637D919F92E9D B8FB2DFC213A88D21C9DC8DCBA917D81 70511DE1BADD360D50058F794B0960AE 11FA28D392CFF907A62D13E3357B1DC0 B51BE089D0B682863B2217201E73A1A9 031968A9B4121DCBC3281A69739AF874 29F5B3AC5471E7B6A04A2C0F2F8A25FD 772A317DF97FC5463FEAC248EB8AB8BE	128
<b>ciphered-content</b>		0
<b>length</b>	81EB	2
<b>value</b>	3100000000F435069679270C5BF4425E E5777402A6C8D51C620EED52DBB18837 8B836E2857D5C053E6DDF27FA87409AE F502CD9618AE47017C010224FD109CC0 BEB21E742D44AB40CD11908743EC90EC 8C40E221D517F72228E1A26E827F43DC 18ED27B5F458D66508B05A2A4CC6FED1 78C881AFC3BC67064689BE8BB41C80AB B3C114A31F4CB03B8B64C7E0B4CE77B2 399C93347858888F92239713B38DF01C 4858245827A92EF334172EA636B31CBB DF2A96AD5D035F66AA38F1A2D97D4BBA 99622E6B5F18789CECB2DFB3937D9F3E 17F8B472098E6563238F375283748098 36002AEA6E7012D2ADFAA7	235
ACCESS.request with authenticated encryption SC II IC II ciphertext II auth. Tag		

<b>general-ciphering(encoded)</b>	DD080102030405060708084D4D4D0000 BC614E084D4D4D000000000100000102 01018180C323C2BD45711DE4688637D9 19F92E9DB8FB2DFC213A88D21C9DC8DC BA917D8170511DE1BADB360D50058F79 4B0960AE11FA28D392CFF907A62D13E3 357B1DC0B51BE089D0B682863B221720 1E73A1A9031968A9B4121DCBC3281A69 739AF87429F5B3AC5471E7B6A04A2C0F 2F8A25FD772A317DF97FC5463FEAC248 EB8AB8BE81EB3100000000F435069679 270C5BF4425EE5777402A6C8D51C620E ED52DBB188378B836E2857D5C053E6DD F27FA87409AEF502CD9618AE47017C01 0224FD109CC0BEB21E742D44AB40CD11 908743EC90EC8C40E221D517F72228E1 A26E827F43DC18ED27B5F458D66508B0 5A2A4CC6FED178C881AFC3BC67064689 BE8BB41C80ABB3C114A31F4CB03B8B64 C7E0B4CE77B2399C93347858888F9223 9713B38DF01C4858245827A92EF33417 2EA636B31CBDF2A96AD5D035F66AA38 F1A2D97D4BBA99622E6B5F18789CECB2 DFB3937D9F3E17F8B472098E6563238F 37528374809836002AEA6E7012D2ADFA A7	401
<b>General-Ciphering</b>	DD	1
<b>transaction-id</b>		0
<b>length</b>	08	1
<b>value</b>	0123456789012345	8
<b>originator-system-title</b>		0
<b>length</b>	08	1
<b>value</b>	4D4D4D0000000001	8
<b>recipient-system-title</b>		0
<b>length</b>	08	1
<b>value</b>	4D4D4D0000BC614E	8
<b>date-time OPTIONAL</b>		0
<b>length</b>	00	1
<b>value</b>		0
<b>other-information</b>		0
<b>length</b>	00	1
<b>value</b>		0
<b>key-info OPTIONAL</b>	01	1
<b>agreed-key CHOICE</b>	02	1
<b>key-parameters</b>		0
<b>length</b>	01	1
<b>value</b>	01	1
<b>key-ciphered-data</b>		0
<b>length</b>	8180	2
<b>value</b>	6439724714B47CD9CB988897D8424AB9 46DCD083D37A954637616011B9C23787 73295F0F850D8DAFD1B8E9FE666E53E4 F097CD10B38B69622152724A90987444 E1FF47974A1F6931A6502F58147463F0 E8CC517D47F55B0AC56DD8AC5C9D0E48 1934F2D90F9893016BD82B6E3FFE21FF 1588F3278B4E9D98EB4FB62ADD64B380	128

<b>ciphered-content</b>		0
<b>length</b>	3D	1
<b>value</b>		
ACCESS.response with authenticated encryption SC II IC II ciphertext II auth. tag	3100000000B3FFCAA594642D8319CEC6 B2A233E2BF4621D6991B97E4565B986E 8CCBE9A299D8E7869723638FF6BB20E6 6E175E6F2D762CFD26B3D58733	61
<b>general-ciphering (encoded)</b>	DD080123456789012345084D4D4D0000 000001084D4D4D0000BC614E00000102 010181806439724714B47CD9CB98897 D8424AB946DCD083D37A954637616011 B9C2378773295F0F850D8DAFD1B8E9FE 666E53E4F097CD10B38B69622152724A 90987444E1F47974A1F6931A6502F58 147463F0E8CC517D47F55B0AC56DD8AC 5C9D0E481934F2D90F9893016BD82B6E 3FFE21FF1588F3278B4E9D98EB4FB62A DD64B3803D31000000000B3FFCAA59464 2D8319CEC6B2A233E2BF4621D6991B97 E4565B986E8CCBE9A299D8E786972363 8FF6BB20E66E175E6F2D762CFD26B3D5 8733	226

### 9.2.7.2.5 Digital signature

The algorithm is the elliptic curve digital signature algorithm (ECDSA) as specified in 9.2.3.4.5.

The structure of the general-signing APDU is shown in Figure 110. For the additional fields, see Table 47 and 9.3.5.

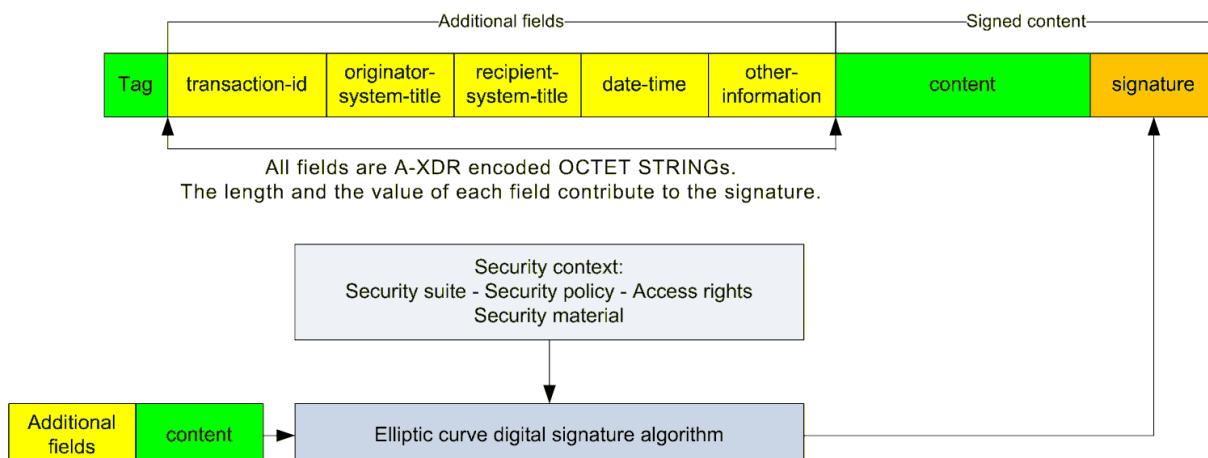


Figure 110 – Structure of general-signing APDUs

### 9.2.7.3 Multi-layer protection by multiple parties

Cryptographic protection can be applied by multiple parties. Generally the parties are:

- a server;
- a client;
- a third party.

Each party can apply one or multiple layers of protection:

- to apply encryption, authentication or authenticated encryption, the ciphering APDUs are used. A third party shall use the general-ciphering APDU. The client can use any of the ciphering APDUs. Authenticated encryption is considered to be a single layer of protection;
- to apply digital signature, the general-signing APDU is used.

If both ciphering and digital signature is applied by the same party for the same party, then normally the digital signature is applied first.

Both the general-ciphering and general-signing APDUs may include the Originator\_System\_Title and the Recipient\_System\_Title. This allows the party that applied the protection to be identified and the party that shall check / remove the protection.

The protection to be applied on the response depends on the security policy and the access rights on the response and on the protection applied on the request. If a kind of protection has been applied on the request by a party, then the same kind of protection will be applied for the same party in the response. However if a kind of protection which was applied on the request is not required on the response, than no protection will be applied on the response for that party.

**Example 1** If the request was digitally signed by the third party and authenticated by the client, and the required protection on the response is authentication and digital signature, then the response will be authenticated for the client and digitally signed for the third party.

**Example 2** If the request was digitally signed by the third party and authenticated by the client, and the required protection on the response is authentication only, then the response will be authenticated for the client and no protection will be applied to the third party. (The TP will receive a general-ciphering APDU without any protection applied.)

If a protection is required on the response that was not applied on the request, then the server cannot determine from the request which party has the response to be protected for. Therefore it shall apply the protection for all parties. See also Annex D.

#### 9.2.7.4 HLS authentication mechanisms

HLS authentication requires cryptographic processing of the challenges exchanged by the client and the server. The HLS authentication mechanisms, the information exchanged and the formulae to process the challenges are shown in Table 50.

**Table 50 – DLMS/COSEM HLS authentication mechanisms**

Authentication mechanism	Pass 1: C → S	Pass 2: S → C	Pass 3: C → S f(StoC)	Pass 4: S → C f(CtoS)
	Carried by			
	AARQ	AARE	XX.request reply_to_HLS authentication	XX.response reply_to_HLS authentication
mechanism_id(2) HLS man. Spec.			Man. Spec.	Man. Spec.
mechanism_id(3) HLS MD5 <sup>1</sup>	CtoS: Random string 8-64 octets	StoC: Random string 8-64 octets	MD5(StoC    HLS Secret)	MD5(CtoS    HLS Secret)
mechanism_id(4) HLS SHA-1 <sup>1</sup>			SHA-1(StoC    HLS Secret)	SHA-1(CtoS    HLS Secret)
mechanism_id(5) HLS GMAC	CtoS: Random string 8-64 octets	StoC: Random string 8-64 octets	SC II IC II GMAC (SC    AK    StoC)	SC II IC II GMAC (SC    AK    CtoS)
mechanism_id(6) HLS SHA-256	Optionally: System-Title-C in calling-AP-title	Optionally: System-Title-S in responding-AP-title	SHA-256 (HLS_Secret    SystemTitle-C    SystemTitle-S    StoC    CtoS)	SHA-256 (HLS_Secret    SystemTitle-S    SystemTitle-C    CtoS    StoC)
mechanism_id(7) HLS ECDSA	CtoS: Random string 32 to 64 octets Optionally: System-Title-C in calling-AP-title, Cert-Sign-Client in calling-AE-qualifier	StoC: Random string 32 to 64 octets Optionally: System-Title-S in responding-AP-title, Cert-Sign-Server responding-AE- qualifier	ECDSA( SystemTitle-C    SystemTitle-S    StoC    CtoS)	ECDSA( SystemTitle-S    SystemTitle-C    CtoS    StoC)

Authentication mechanism	Pass 1: C → S	Pass 2: S → C	Pass 3: C → S f(StoC)	Pass 4: S → C f(CtoS)
Legend:				
<ul style="list-style-type: none"> <li>- C: Client, S: Server, CtoS: Challenge client to server, StoC: Challenge server to client</li> <li>- IC: Invocation counter</li> <li>- xx.request / .response: xDLMS service primitives used to access the <i>reply_to_HLS authentication</i> method of the “Association SN / LN” object.</li> </ul>				
<sup>1</sup> The use of authentication mechanisms 3 and 4 are not recommended for new implementations.				
NOTE The system titles and the Certificates have to be sent only if not already known by the other party.				

Where the system titles and the certificates for the digital signature key are also needed, these may be transported in the AARQ / AARE APDUs, carrying the COSEM-OPEN service .request / .response, see Figure 89. The System\_Title and Cert-Sign may be already known; in this case they do not have to be transported. If these elements are not available, the result of the processing of the challenge fails and the AA shall not be established.

Table 51 provides a test vector for HLS authentication-mechanism 5 with GMAC.

**Table 51 – HLS example using authentication-mechanism5 with GMAC**

Security material	X	Contents	LEN(X) bytes	len(X) bits						
Security suite		GCM-AES-128								
System Title	Sys-T	<table border="1"> <tr> <td>Client</td> <td>Server</td> </tr> <tr> <td><b>4D4D4D0000000001</b></td> <td><b>4D4D4D0000BC614E</b></td> </tr> </table> <p>(here, the five last octets contain the manufacturing number in hexa)</p>	Client	Server	<b>4D4D4D0000000001</b>	<b>4D4D4D0000BC614E</b>				
Client	Server									
<b>4D4D4D0000000001</b>	<b>4D4D4D0000BC614E</b>									
Invocation counter	IC	<table border="1"> <tr> <td>00000001</td> <td>01234567</td> </tr> </table>	00000001	01234567	4	32				
00000001	01234567									
Initialization Vector	IV	<table border="1"> <tr> <td colspan="2">Sys-T II IC</td> </tr> <tr> <td>Client</td> <td>Server</td> </tr> <tr> <td><b>4D4D4D0000000001</b> 00000001</td> <td><b>4D4D4D0000BC614E</b> 01234567</td> </tr> </table>	Sys-T II IC		Client	Server	<b>4D4D4D0000000001</b> 00000001	<b>4D4D4D0000BC614E</b> 01234567	12	96
Sys-T II IC										
Client	Server									
<b>4D4D4D0000000001</b> 00000001	<b>4D4D4D0000BC614E</b> 01234567									
Block cipher key (global)	EK	000102030405060708090A0B0C0D0E0F	16	128						
Authentication Key	AK	D0D1D2D3D4D5D6D7D8D9DADBCDDDEF	16	128						
Security control byte	SC	10	1	8						
		<b>Pass 1: Client sends challenge to server</b>								
CtoS		4B35366956616759 "K56iVagY"	8	64						
		<b>Pass 2: Server sends challenge to client</b>								
StoC		503677524A323146 "P6wRJ21F"	8	64						
		<b>Pass 3: Client processes StoC</b>								
SC II AK II StoC		10D0D1D2D3D4D5D6D7D8D9DADBCDDDEF5036775 24A323146								
T = GMAC (SC II AK II StoC)		1A52FE7DD3E72748973C1E28	12	96						
f(StoC) = SC II IC II T		10000000011A52FE7DD3E72748973C1E28	17	136						
		<b>Pass 4: Server processes CtoS</b>								
SC II AK II CtoS		10D0D1D2D3D4D5D6D7D8D9DADBCDDDEF4B35366 956616759								
T (SC II AK II CtoS)		FE1466AFB3DBCD4F9389E2B7	12	96						
f(CtoS) = SC II IC II T		1001234567FE1466AFB3DBCD4F9389E2B7	17	136						

Table 52 provides a test vector for HLS authentication-mechanism 7 with ECDSA.

**Table 52 – HLS example using authentication-mechanism 7 with ECDSA**

Security Material	X	Contents	LEN (Bytes)
Security Suite		ECDH-ECDSA-AES-128-GCM	
Curve		P-256	
Domain Parameters	D	See Table A. 1.	
System Title Client	Sys-TC	4D4D4D0000BC614E	8
System Title Server	Sys-TS	4D4D4D0000000001	8
Private Key Client	Pri-KC	E9A045346B2057F1820318AB125493E9AB36CE5 90011C0FF30090858A118DD2E	32
Private Key Server	Pri-KS	B582D8C910018302BA3131BAB9BB6838108BB94 08C30B2E49285985256A59038	32
Public Key Client	Pub-KC	917DBFECA43307375247989F07CC23F53D4B963 AF8026C749DB33852011056DFDBE8327BD69CC1 49F018A8E446DDA6C55BCD78E596A56D4032362 33F93CC89B3	64
Public Key Server	Pub-KS	E4D07CEB0A5A6DA9D2228B054A1F5E295E1747A 963974AF75091A0B0BC2FB92DA7D2ABD9FDD415 79F36A1C8171A0CB638221DF1949FD95C8FAE14 8896920450D	64
Challenge Client To Server	CtoS	2CA1FC2DE9CD03B5E8E234CEA16F2853F6DC5F5 4526F4F4995772A50FB7E63B3	32
Challenge Server To Client	StoC	18E95FFE3AD0DCABDC5D0D141DC987E270CB0A3 95948D4231B09DE6579883657	32
ECDSA(SystemTitle-C    SystemTitle-S    StoC    CtoS) (calculated with Pri-KC)	f(StoC)	C5C6D6620BDB1A39FCE50F4D64F0DB712D6FB57 A64030B0C297E1250DC859660D3B1FA334AD804 11807369F5DD3BC17B59894C9E9C11C59376580 D15A2646D16	64
ECDSA(SystemTitle-S    SystemTitle-C    CtoS    StoC) (calculated with Pri-KS)	f(CtoS)	946C2E3E4F18291571F4A45ACB7086100574694 A3BAF67D2D147FE8F92481A5AB2186C5CBC3F80 E94482D9388B85C6A73E5FD687F09773C1F615A A2A905ED057	64
NOTE The values of the public keys are represented here as FE2OS( $x_p$ )II FE2OS( $y_p$ ).			

### 9.2.7.5 Protecting COSEM data

The cryptographic algorithms applied to xDLMS APDUs can be also applied to COSEM data, i.e. attribute values and method invocation / return parameters. This is achieved by accessing attributes and/or methods of other COSEM objects indirectly through instances of the “Data protection” interface class, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.9.

The list of data to be protected, the required protection and the protection parameters are determined by the “Data protection” objects.

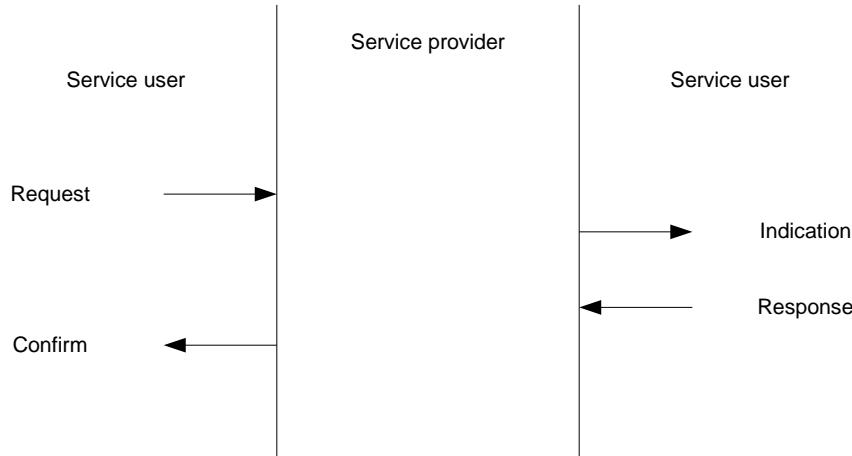
“Data protection” objects allow applying or removing protection when reading or writing a list of attributes, or when invoking methods of COSEM objects. Protection to be applied / removed may include any combination of authentication, encryption and digital signature.

The APDUs carrying the service invocations to access the attributes and methods of “Data protection” objects are protected as required by the prevailing security policy and the access rights of the “Data protection” object.

### 9.3 DLMS/COSEM application layer service specification

#### 9.3.1 Service primitives and parameters

In general, the services of a layer (or sublayer) are the capabilities it offers to a user in the next higher layer (or sublayer). In order to provide its service, a layer builds its functions on the services it requires from the next lower layer. Figure 111 illustrates this notion of service hierarchy and shows the relationship of the two correspondent N-users and their associated N-layer peer protocol entities.



**Figure 111 – Service primitives**

Services are specified by describing the information flow between the N-user and the N-layer. This information flow is modelled by discrete, instantaneous events, which characterize the provision of a service. Each event consists of passing a service primitive from one layer to the other through an N-layer service access point associated with an N-user. Service primitives convey the information required in providing a particular service. These service primitives are an abstraction in that they specify only the service provided rather than the means by which the service is provided. This definition of service is independent of any particular interface implementation.

Services are specified by describing the service primitives and parameters that characterize each service. A service may have one or more related primitives that constitute the activity that is related to the particular service. Each service primitive may have zero or more parameters that convey the information required to provide the service. Primitives are of four generic types:

- REQUEST: The request primitive is passed from the N-user to the N-layer to request that a service be initiated;
- INDICATION: The indication primitive is passed from the N-layer to the N-user to indicate an internal N-layer event that is significant to the N-user. This event may be logically related to a remote service request, or may be caused by an event internal to the N-layer;
- RESPONSE: The response primitive is passed from the N-user to the N-layer to complete a procedure previously invoked by an indication primitive;
- CONFIRM: The confirm primitive is passed from the N-layer to the N-user to convey the results of one or more associated previous service request(s).

Possible relationships among primitive types are illustrated by the time-sequence diagrams shown in Figure 112. The figure also indicates the logical relationship of the primitive types. Primitive types that occur earlier in time and are connected by dotted lines in the diagrams are the logical antecedents of subsequent primitive types.

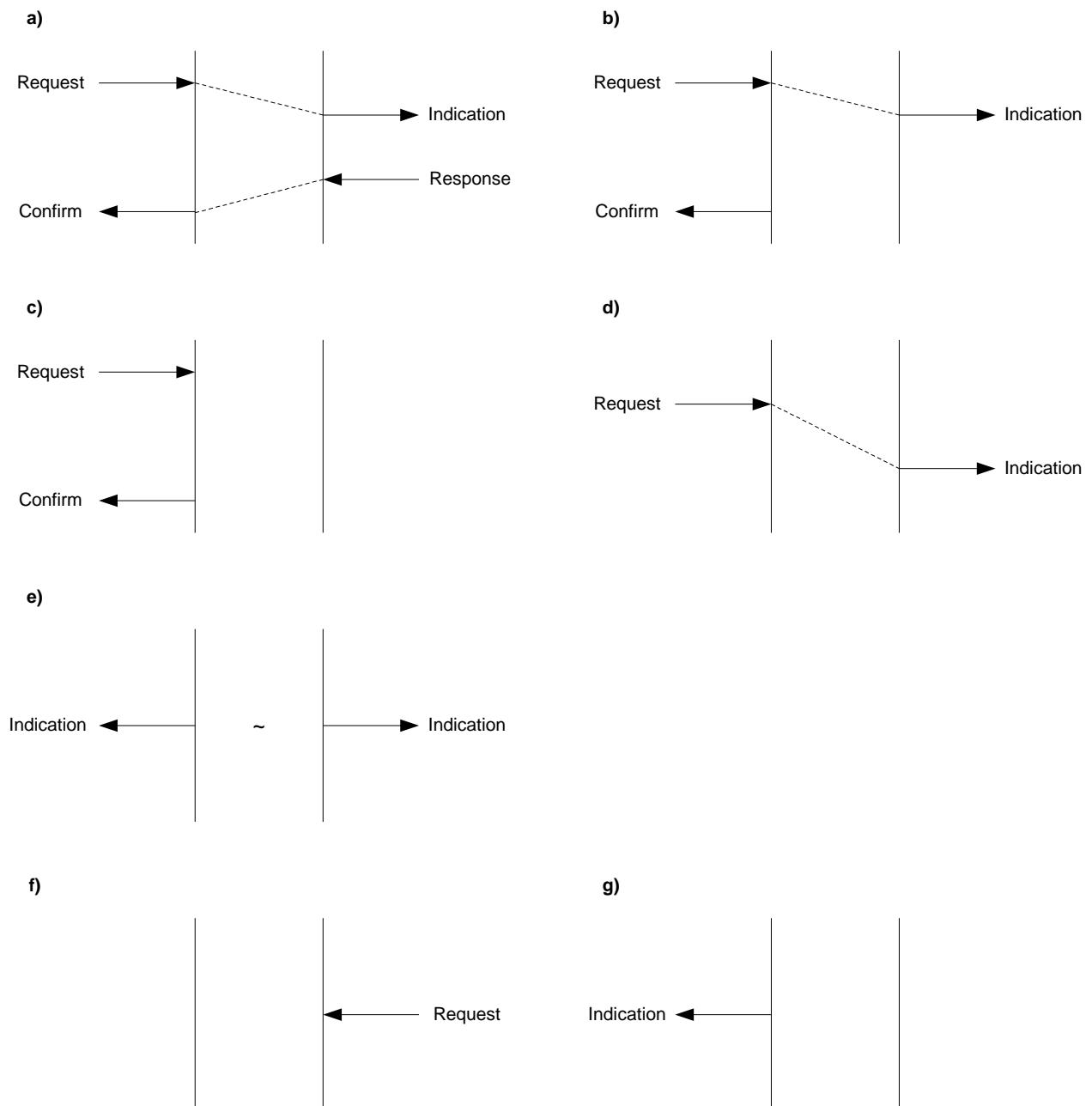


Figure 112 – Time sequence diagrams

The service parameters of the DLMS/COSEM AL service primitives are presented in a tabular format. Each table consists of two to five columns describing the service primitives and their parameters. In each table, one parameter – or a part of it – is listed on each line. In the appropriate service primitive columns, a code is used to specify the type of usage of the parameter. The codes used are listed in Table 53.

Some parameters may contain sub-parameters. These are indicated by labelling of the parameters as M, U, S or C, and indenting all sub-parameters under the parameter. Presence of the sub-parameters is always dependent on the presence of the parameter that they appear under. For example, an optional parameter may have sub-parameters; if the parameter is not supplied, then no sub-parameters may be supplied.

**Table 53 – Codes for AL service parameters**

M	The parameter is mandatory for the primitive.
U	The parameter is a user option, and may or may not be provided depending on dynamic usage by the ASE user.
S	The parameter is selected among other S-parameters as internal response of the server ASE environment.
C	The parameter is conditional upon other parameters or the environment of the ASE user.
(–)	The parameter is never present.
=	The " (=) " code following one of the M, U, S or C codes indicates that the parameter is semantically equivalent to the parameter in the service primitive to its immediate left in the table. For instance, an " M (=) " code in the .indication service primitive column and an "M" in the .request service primitive column means that the parameter in the .indication primitive is semantically equivalent to the one in the .request primitive.

Throughout this Technical Report, the following rules are observed regarding the naming of terms:

- the name of ACSE services and the data transfer services using LN referencing is written in uppercase. Examples are: COSEM-OPEN, GET;
- the name of the data transfer services using SN referencing is written in title case. Examples are: Read, Write;
- camel notation is used in the following cases: DataNotification, EventNotification, TriggerEventNotificationSending, UnconfirmedWrite, InformationReport;
- the types of the LN service primitives may be mentioned in two alternative forms. Examples: “GET.request service primitive of Request\_Type == NORMAL” or “GET-REQUEST-NORMAL service primitive”;
- service parameter name elements are capitalized and joined with an underscore to signify a single entity: Examples are Protocol\_Connection\_Parameters and COSEM\_Attribute\_Descriptor;
- when the same parameter may occur several times, this is indicated by repeating the parameter in curly brackets. Example: Data { Data };
- in the data transfer service specifications, parameters used with block transfer only are shown in bold. Example: **DataBlock\_G**;

NOTE This applies only to the service-specific block transfer mechanism.

- direct reference to a service parameter uses the capitalized form, while indirect (non-specific) reference uses the small caps without underscore joining. A direct reference example is: “The COSEM\_Attribute\_Descriptor parameter references a COSEM object attribute.” An indirect (non-specific) reference example is: “A GET-REQUEST-NORMAL service primitive contains a single COSEM attribute descriptor”;
- the names of COSEM data transfer APDUs using LN referencing are capitalized and joined with a dash to signify a single entity. Example: Get-Request-Normal;
- the names of COSEM data transfer APDUs using SN referencing use the camel notation. Example: ReadRequest.

### 9.3.2 The COSEM-OPEN service

#### 9.3.2.1 Function

The function of the COSEM-OPEN service is to establish an AA between peer COSEM Application Entities (AEs). It uses the A-ASSOCIATE service of the ACSE. The COSEM-OPEN service provides only the framework for *transporting* this information. To provide and verify that information is the job of the appropriate COSEM AP.

#### 9.3.2.2 Semantics of the service primitives

The COSEM-OPEN service primitives shall provide parameters as shown in Table 54.

**Table 54 – Service parameters of the COSEM-OPEN service primitives**

	.request	.indication	.response	.confirm
Protocol_Connection_Parameters	M	M (=)	M	M (=)
ACSE_Protocol_Version	U	U (=)	U	U (=)
Application_Context_Name	M	M (=)	M	M (=)
Called_AP_Title	U	U (=)	–	–
Called_AE_Qualifier	U	U(=)	–	–
Called_AP_Invocation_Identifier	U	U (=)	–	–
Called_AE_Invocation_Identifier	U	U (=)	–	–
Calling_AP_Title	C	C (=)	–	–
Calling_AE_Qualifier	U	U (=)	–	–
Calling_AP_Invocation_Identifier	U	U (=)	–	–
Calling_AE_Invocation_Identifier	U	U (=)	–	–
Local_Or_Remote	–	–	–	M
Result	–	–	M	M
Failure_Type	–	–	M	M
Responding_AP_Title	–	–	C	C (=)
Responding_AE_Qualifier	–	–	U	U (=)
Responding_AP_Invocation_Identifier	–	–	U	U (=)
Responding_AE_Invocation_Identifier	–	–	U	U (=)
ACSE_Requirements	U	U (=)	U	U (=)
Security_Mechanism_Name	C	C (=)	C	C (=)
Calling_Authentication_Value	C	C (=)	–	–
Responding_Authentication_Value	–	–	C	C (=)
Implementation_Information	U	U (=)	U	U (=)
Proposed_xDLMS_Context	M	M (=)	–	–
Dedicated_Key	C	C (=)	–	–
Response_Allowed	C	C (=)	–	–
Proposed_DLMS_Version_Number	M	M (=)	–	–
Proposed_DLMS_Conformance	M	M (=)	–	–
Client_Max_Receive_PDU_Size	M	M (=)	–	–
Negotiated_xDLMS_Context	–	–	S	S (=)
Negotiated_DLMS_Version_Number	–	–	M	M (=)
Negotiated_DLMS_Conformance	–	–	M	M (=)
Server_Max_Receive_PDU_Size	–	–	M	M (=)
VAA_Name			M	M (=)
xDLMS_Initiate_Error			S	S (=)
User_Information	U	C (=)	–	–
Service_Class	M	M (=)	–	–

The service parameters of the COSEM-OPEN.request service primitive, except the Protocol\_Connection\_Parameters, the User\_Information parameter and – depending on the communication profile – the Service\_Class parameter are carried by the fields of the AARQ APDU sent by the client.

The service parameters of the COSEM-OPEN.response service primitive, except the Protocol\_Connection\_Parameters is carried by the fields of the AARE APDU sent by the server.

The A-ASSOCIATE service and the AARQ and AARE APDUs are specified in 9.4.2. Encoding examples are given in 11.

The Protocol\_Connection\_Parameters parameter is mandatory. It contains all information necessary to use the layers of the communication profile, including the communication profile (protocol) identifier and the addresses required. It identifies the participants of the AA. The elements of this parameter are passed to the entities managing lower layer connections and to the lower layers as appropriate.

The ACSE\_Protocol\_Version parameter is optional. If present, the default value shall be used.

The Application\_Context\_Name parameter is mandatory. In the request primitive, it holds the value proposed by the client. In the response primitive, it holds the same value or the value supported by the server.

The use of the Called\_AP\_Title, Called\_AE\_Qualifier, Called\_AP\_Invocation\_Identifier, Called\_AE\_Invocation\_Identifier parameters is optional. Their use is not specified in this Technical Report.

The use of the Calling\_AP\_Title parameter is conditional. When the proposed application context and/or the proposed HLS authentication mechanism require the use of the client system title and it has not yet been transferred during the registration process then the Calling\_AP\_Title shall carry the client system title. See also 4.3.4.

The use of the Calling\_AE\_Qualifier parameter is conditional. When the Application\_Context\_Name indicates an application context using ciphering, it may carry the public digital signature key certificate of the client.

The use of the Calling\_AP\_Invocation\_Identifier is optional. Its use is not specified in this Technical Report.

The use of the Calling\_AE\_Invocation\_Identifier parameter is optional. When present, it carries the identifier of the client-side user of the AA.

NOTE 1 The client user identification mechanism is specified in DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.2.

The Local\_or\_Remote parameter is mandatory. It indicates the origin of the COSEM-OPEN.confirm primitive. It is set to Remote if the primitive has been generated following the reception of an AARE APDU from the server. It is set to Local if the primitive has been locally generated.

The Result parameter is mandatory. In the case of remote confirmation, it indicates whether the Server accepted the proposed AA or not. In the case of local confirmation, it indicates whether the client side protocol stack accepted the request or not.

The Failure\_Type parameter is mandatory. In the case of remote confirmation, it carries the information provided by the server. In the case of local and negative confirmation, it indicates the reason for the failure.

The use of the Responding\_AP\_Title parameter is conditional. When the negotiated application context and/or the negotiated HLS authentication mechanism require the use of the server system title and it has not yet been transferred during the registration process then the Responding\_AP\_Title shall carry the server system title. See also 4.3.4.

The use of the Responding\_AE\_Qualifier is conditional. When the Application\_Context\_Name indicates an application context using ciphering, it may carry the public digital signature key certificate of the server.

The use of the Responding\_AP\_Invocation\_Identifier and Responding\_AE\_Invocation\_Identifier parameters is optional. Their use is not specified in this Technical Report.

The ACSE\_Requirements parameter is optional. It is used to select the optional authentication functional unit of the A-Associate service for the association. See 9.4.2.1.

The presence of the ACSE\_Requirements parameter depends on the authentication mechanism used:

- in the case of Lowest Level Security authentication it shall not be present; only the Kernel functional unit is used;
- in the case of Low Level Security (LLS) authentication it shall be present in the .request primitive and it may be present in the .response service primitive and it shall indicate authentication (bit 0 set);
- in the case of High Level Security (HLS) authentication, it shall be present both in the .request and the .response service primitives and it shall indicate authentication (bit 0 set).

The Security\_Mechanism\_Name parameter is conditional. It is present only, if the authentication functional unit has been selected. If present, the .request primitive holds the value proposed by the client and the .response primitive holds the value required by the server, i.e. the one to be used by the client.

The Calling.Authentication\_Value parameter and the Responding.Authentication\_Value parameters are conditional. They are present only, if the authentication functional unit has been selected. They hold the client authentication value / server authentication value respectively, appropriate for the Security\_Mechanism\_Name.

The Implementation\_Information parameter is optional. Its use is not specified in this Technical Report.

The Proposed\_xDLMS\_Context parameter holds the elements of the proposed xDLMS context. It is carried by the xDLMS InitiateRequest APDU, placed in the user-information field of the AARQ APDU.

The Dedicated\_Key element is conditional. It may only be present when the Application\_Context\_Name parameter indicates an application context using ciphering. The dedicated key is used for dedicated ciphering of xDLMS APDUs exchanged within the AA established.

**When the dedicated key is not present, the xDLMS InitiateRequest APDU does not have to be protected.**

**When the dedicated key is present, the xDLMS InitiateRequest APDU shall be protected, i.e. at least authenticated and encrypted using the AES-GCM algorithm, the global unicast encryption key and the authentication key (if in use). In addition, it shall also be digitally signed if required by the security policy.**

The xDLMS InitiateRequest APDU shall be protected the same way as described above, when the dedicated key is not present, but it is necessary to protect the RLRQ APDU by including the protected xDLMS InitiateRequest in its user-information field. See 9.3.3.

The use of Response\_Allowed element is conditional. It indicates if the server is allowed to respond with an AARE APDU, i.e. if the AA to be established is confirmed (Response\_Allowed == TRUE) or not confirmed (Response\_Allowed == FALSE).

The Proposed\_DLMS\_Version\_Number element holds the proposed DLMS version number. See 9.1.4.

The Proposed\_DLMS\_Conformance element holds the proposed conformance block. See 9.4.6.1.

The Client\_Max\_Receive\_PDU\_Size element holds the maximum length of the xDLMS APDUs the client can receive. See Table 20.

If the xDLMS context proposed by the client is acceptable for the server, then the response service primitive shall contain the Negotiated\_xDLMS\_Context parameter. It is carried by the xDLMS InitiateResponse APDU, placed in the user-information field of the AARE APDU. If the xDLMS InitiateRequest APDU has been ciphered, the xDLMS InitiateResponse APDU shall be also ciphered the same way.

The Negotiated\_DLMS\_Version\_Number element holds the negotiated DLMS version number. See 9.1.4.

The Negotiated\_DLMS\_Conformance element holds the negotiated conformance block. See 9.4.6.1.

The Server\_Max\_Receive\_PDU\_Size element carries the maximum length of the xDLMS APDUs the server can receive. See Table 20.

The VAA\_name element carries the dummy value of 0x0007 in the case of LN referencing, and the base\_name of the current Association object, 0xFA00, in the case of SN referencing.

If the xDLMS context proposed by the client is not acceptable for the server, then the response service primitive shall carry the xDLMS\_Initiate\_Error parameter. It is carried by the confirmedServiceError APDU, with appropriate diagnostic elements, placed in the user-information field of the AARE APDU.

The User\_Information parameter is optional. If present, it shall be passed on to the supporting protocol layer, provided it is capable to carry it. The .indication primitive shall then contain the user-specific information carried by the supporting lower protocol layer(s). See Clause 10.

NOTE 2 The User\_Information parameter of the COSEM-OPEN service is not to be confused with the user-information field of the AARQ / AARE APDUs.

The Service\_Class parameter is mandatory. It indicates whether the service is invoked in a confirmed or in an unconfirmed manner. The handling of this parameter may depend on the communication profile. See Clause 10.

### 9.3.2.3 Use

Possible logical sequences of the COSEM-OPEN service primitives are illustrated in Figure 112.

- for confirmed AA – successful or unsuccessful – establishment, item a);
- for unconfirmed AA establishment, item b);
- in the case of a pre-established AA or an unsuccessful attempt due to a local error, item c).

The .request primitive is invoked by the client AP to request the establishment of a confirmed or an unconfirmed AA with a server AP.

NOTE 3 Before the invocation of the COSEM-OPEN.request primitive, the physical layers have to be connected. Depending on the communication profile, the invocation of this primitive may also imply the connection of other lower layers.

Upon the reception of the request invocation, the AL constructs and sends an AARQ APDU to the server.

The .indication primitive is generated by the server AL when a correctly formatted AARQ APDU is received.

The .response primitive is invoked by the server AP to indicate to the AL whether the proposed AA is accepted or not. It is invoked only if the proposed AA is confirmed. The AL constructs then an AARE APDU and sends it to its peer, containing the service parameters received from the AP.

The .confirm primitive is generated by the client AL to indicate to the client AP whether the AA requested previously is accepted or not:

- remotely, when an AARE APDU is received;
- locally, if the requested AA already exists; this includes pre-established AAs;
- locally, if the corresponding .request primitive has been invoked with Service\_Class == Unconfirmed;
- locally, if the requested AA is not allowed;
- locally, if an error is detected: missing or not correct parameters, failure during the establishment of the requested lower layer connections, missing physical connection, etc.

The protocol for establishing an AA is specified in 9.4.4. Communication profile specific rules are specified in Clause 10.

### 9.3.3 The COSEM-RELEASE service

#### 9.3.3.1 Function

The function of the COSEM-RELEASE service is to gracefully release an existing AA. The way it is invoked determines whether or not it uses the A-RELEASE service of the ACSE.

#### 9.3.3.2 Semantics of the service primitives

The COSEM-RELEASE service primitives shall provide parameters as shown in Table 55.

**Table 55 – Service parameters of the COSEM-RELEASE service primitives**

	.request	.indication	.response	.confirm
Use_RLRQ_RLRE	U	C(=)	C(=)	-
Reason	U	U (=)	U	U (=)
Proposed_xDLMS_Context	C	C (=)	-	-
Negotiated_xDLMS_Context	-	-	C	C (=)
Local_Or_Remote	-	-	-	M
Result	-	-	M	M
Failure_Type	-	-	-	C
User_Information	U	C (=)	U	C (=)

Where:

- Use\_RLRQ\_RLRE in the .request primitive is optional.

If present, its value may be TRUE or FALSE (default is FALSE). It indicates whether or not the ACSE A-RELEASE service should be used:

In .request:

If TRUE: The A-RELEASE service should be used and this would involve the use of an RLRQ/RLRE APDU exchange. The A\_RELEASE service and RLRQ / RLRE APDUs are specified in 9.4.2.

In .response:

The Use\_RLRQ\_RLRE parameter in the .response primitive is conditional. If it was present in the .indication primitive and its value was TRUE, it shall also be present and its value shall be TRUE. If Use\_RLRQ\_RLRE in .request is FALSE then it shall not be present in .response or its value shall be FALSE.

If the value of the Use\_RLRQ\_RLRE parameter is FALSE, then the AA can be released by disconnecting the supporting protocol layer of the AL.

- Reason is optional.

It may be present only if the value of the Use\_RLRQ\_RLRE is TRUE. It is carried by the reason field of the RLRQ / RLRE APDU respectively.

When used on the .request primitive, this parameter identifies the general level of urgency of the request. It takes one of the following symbolic values:

- normal;
- urgent (not available in DLMS/COSEM); or
- user defined.

When used on the .response primitive, this parameter identifies information about why the acceptor accepted or rejected the release request. Note, that in DLMS/COSEM the server cannot reject the release request. It takes one of the following symbolic values:

- normal;
- not finished; or
- user defined.

NOTE 1 The value "not finished" is used in the .response primitive when the acceptor is forced to release the association but wishes to give a warning that it has additional information to send or receive.

**Proposed\_xDLMS\_Context** is conditional:

It is present only if the value of the Use\_RLRQ\_RLRE is TRUE and the AA to be released has been established with an application context using ciphering. This option allows the COSEM-RELEASE service to be secured thereby avoiding the possibility of a denial-of-service attack being performed by unauthorized releasing of the AA.

In .request:

Proposed\_xDLMS\_Context shall be the same as in the COSEM-OPEN.request service primitive, having established the AA to be released. It is carried by the xDLMS InitiateRequest APDU, protected the same way as in the AARQ and placed in the user-information field of the RLRQ APDU.

In .response:

If the xDLMS InitiateRequest APDU can be successfully deciphered, then the .response primitive shall carry the same Negotiated\_xDLMS\_Context parameter as in the COSEM-OPEN.response primitive. It is carried by the xDLMS InitiateResponse APDU, protected the same way as in the AARE and placed in the user-information field of the RLRE APDU.

If the xDLMS InitiateRequest APDU cannot be deciphered, the RLRQ APDU is silently discarded.

**Local\_or\_Remote** is mandatory:

It indicates the origin of the COSEM-RELEASE.confirm primitive.

It is set to Remote if either:

- a RLRE APDU has been received from the server; or
- a disconnect confirmation service primitive has been received.

It is set to Local if the primitive has been locally generated.

**Result** is mandatory.

It is used in .response primitive, to indicate whether the server AP can accept the request to release the AA or not. As servers cannot refuse such requests, its value should normally be SUCCESS unless the AA referenced does not exist.

**Failure\_Type** is conditional.

It is present if Result == ERROR. In this case, it indicates the reason for the failure. It is a locally generated parameter on the client side.

**User\_Information** is optional.

In .request:

270/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

If present, it is passed to the supporting protocol layer, provided it is able to carry it. The .indication primitive contains then the user-specific information carried by the supporting lower protocol layer(s).

In .response

If present, it is passed to the supporting protocol layer. In the .confirm primitive, it may be present only when the service is remotely confirmed. It contains then the user-specific information carried by the supporting lower protocol layer(s).

NOTE 2 The User\_Information parameter of the COSEM-RELEASE service is not to be confused with the user-information field of the RLRQ / RLRE APDUs.

The specification of the content of the User\_Information parameter is not within the Scope of this Technical Report. See also Clause 10.

### 9.3.3.3 Use

Possible logical sequences of the COSEM-RELEASE service primitives are illustrated in Figure 112:

- for successful release of a confirmed AA , item a);
- for release of an unconfirmed AA , item b); and
- for an unsuccessful attempt due to a local error item c).

The use of the COSEM-RELEASE service depends on the value of the Use\_RLRQ\_RLRE parameter. When it is invoked with Use\_RLRQ\_RLRE == TRUE, the service is based on the ACSE A-RELEASE service. Otherwise, the invocation of the service leads to the disconnection of the supporting protocol layer.

The .request primitive is invoked by the client AP to request the release of a confirmed or an unconfirmed AA with a server AP. Upon the reception of the request invocation with Use\_RLRQ\_RLRE == TRUE, the AL constructs and sends an RLRQ APDU to the server. Otherwise, it sends an XX-DISCONNECT.request primitive (where XX is the supporting lower protocol layer).

The .indication primitive is generated by the server AL if:

- a RLRQ APDU is received. If a deciphering error occurs, the RLRQ APDU is silently discarded (no .indication primitive is generated); or
- an XX-DISCONNECT.request is received.

The .response primitive is invoked by the server AP, but only if the AA to be released is confirmed. Note, that the server AP cannot refuse this request. Upon the reception of the .response service primitive the server AL:

- sends a RLRE APDU, if the Use\_RLRQ\_RLRE parameter is TRUE; or
- sends an XX-DISCONNECT.response otherwise.

The .confirm primitive is generated by the client AL to indicate to the client AP whether the requested release of the AA is accepted or not:

- remotely, when an XX-DISCONNECT.cnf primitive is received. The supporting protocol layer is disconnected; or
- remotely, when a RLRE APDU is received. The supporting protocol layer is not disconnected; or
- locally, upon the expiry of a time-out on waiting for an RLRE APDU; or
- locally, when an RLRQ APDU to release an unconfirmed AA is sent out; or
- locally, when a local error is detected: missing or not correct parameters, or communication failure at lower protocol layer level etc.

If the RLRE APDU received contains a ciphered xDLMS InitiateResponse APDU but it cannot be deciphered, then the RLRE APDU shall be discarded. It is left to the client to cope with the situation.

The protocol for releasing an AA is specified in 9.4.5. Communication profile specific rules are specified in 10. See also 9.4.2.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	271/614
-----------------------	------------	-----------------------	---------

### 9.3.4 The COSEM-ABORT service

#### 9.3.4.1 Function

The function of the COSEM-ABORT service is to indicate an unsolicited disconnection of the supporting protocol layer.

#### 9.3.4.2 Semantics of the service primitive

The COSEM-ABORT service primitives shall provide parameters as shown in Table 56.

**Table 56 – Service parameters of the COSEM-ABORT service primitives**

<b>.indication</b>	
Diagnostics	U

The Diagnostics parameter is optional. It shall indicate the possible reason for the disconnection, and may carry lower protocol layer dependent information as well. Specification of the contents of this parameter is not within the Scope of this Technical Report.

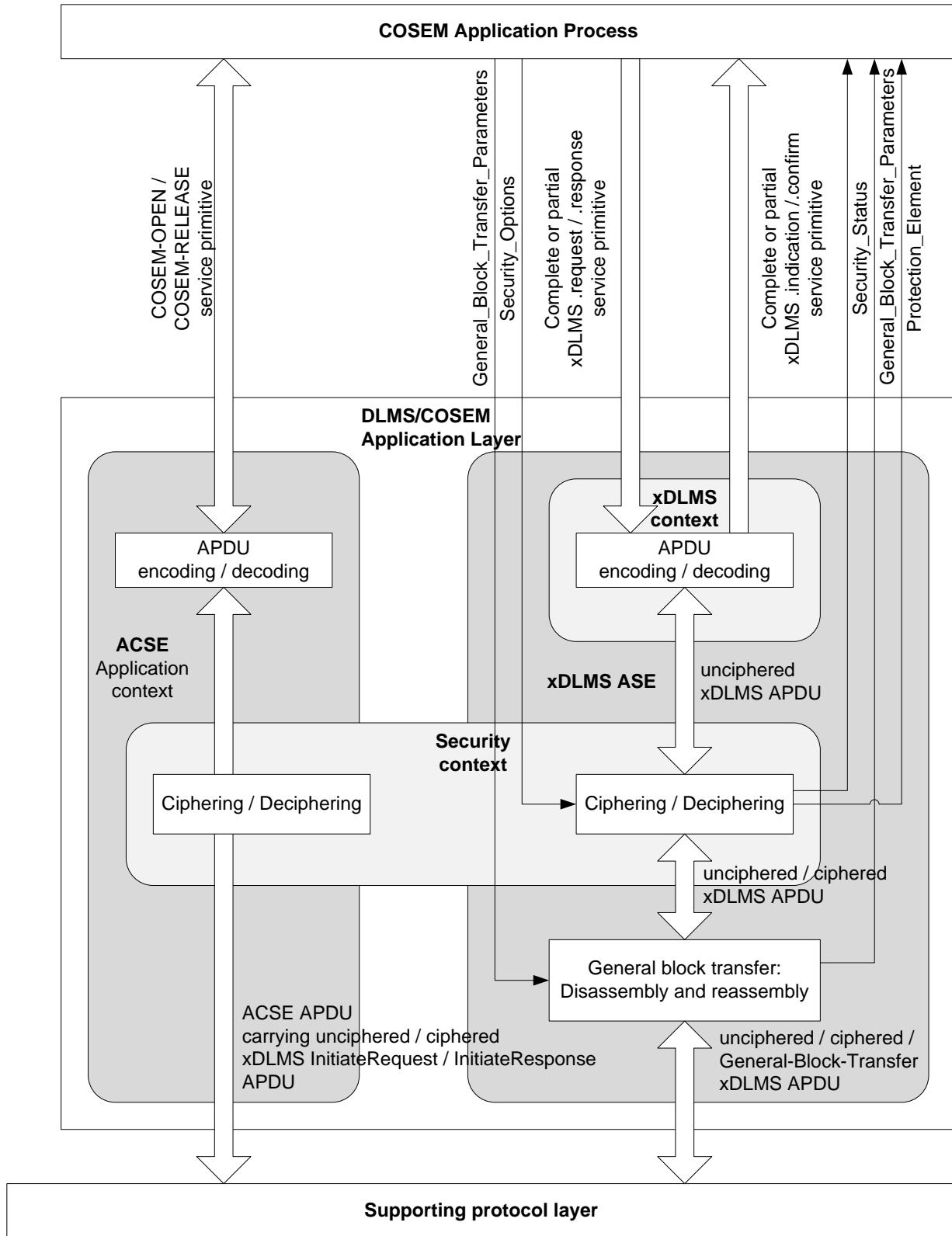
#### 9.3.4.3 Use

The COSEM-ABORT.indication primitive is locally generated both on the client and on the server side to indicate to the COSEM AP that a lower layer connection closed in a non-solicited manner. See Figure 112 item e). The origin of such an event can be an external event (for example the physical line is broken), or an action of a supporting protocol layer connection manager AP, present in some profiles, when the supporting protocol layer connection is not managed by the DLMS/COSEM AL. This shall cause the COSEM APs to abort any existing AAs, except the pre-established ones on the server side.

The protocol for the COSEM-ABORT service is specified in 9.4.5.3.

### 9.3.5 Protection and general block transfer parameters

To control cryptographic protection of xDLMS APDUs and the GBT mechanism, additional service parameters are passed between the AL and the AP as shown in Figure 113 and Table 57.



NOTE For services initiated by the client, the service primitives are .request, .indication, .response and .confirm. For unsolicited services – initiated by the server – the service primitives are .request and .indication.

**Figure 113 – Additional service parameters to control cryptographic protection and GBT**

**Table 57 – Additional service parameters**

	.request	.indication	.response	.confirm
Additional_Service_Parameters	U	–	U (=)	
Invocation_Type	M	–	M (=)	–
Security_Options	C	–	C (=)	–
General_Block_Transfer_Parameters	C	–	C (=)	–
Block_Transfer_Streaming	M	–	M (=)	–
Block_Transfer_Window	M	–	M (=)	–
Service_Parameters	M	–	M (=)	–
Additional_Service_Parameters	–	U	–	U (=)
Invocation_Type	–	U	–	U (=)
Security_Status	–	C	–	C (=)
General_Block_Transfer_Parameters	–	C	–	C (=)
Block_Transfer_Window	–	M	–	M (=)
Service_Parameters	–	M	–	M (=)
Protection_Element	–	C	–	C (=)
NOTE The service primitives available depend on the kind of the service.				

The Additional\_Service\_Parameters are present only if ciphering or GBT is used.

The Invocation\_Type parameter is mandatory: it indicates if the service invocation is complete or partial. Possible values: COMPLETE, FIRST-PART, ONE-PART and LAST-PART.

NOTE 1 Partial service invocations may be useful when the service parameters are long. However, there is no direct relationship between the partial service invocations and the general-block-transfer APDUs.

The Security\_Options parameter is conditional: it is present only if the application context is a ciphered one, the .request / .response service primitive has to be ciphered and Invocation\_Type = COMPLETE or FIRST-PART. It determines the protection to be applied by the AL. See also Table 58, 9.4.6.13 and Figure 139.

The General\_Block\_Transfer\_Parameters parameter is conditional: it is present only if general block transfer (GBT) is used and Invocation\_Type = COMPLETE or FIRST-PART. It provides information on the GBT streaming capabilities:

- the Block\_Transfer\_Streaming parameter is present only in .request and .response service primitives. It is passed by the AP to the AL to indicate if the AL is allowed to send general-block-transfer APDUs using streaming (TRUE) or not (FALSE);
- the Block\_Transfer\_Window parameter indicates the window size supported, i.e. the maximum number of blocks that can be received in a window.

The streaming process itself is managed by the AL. See 9.4.6.13.

The Service\_Parameters are mandatory: they include the parameters of xDLMS service invocations. If Invocation\_Type != COMPLETE, then it includes a part of the service parameters.

The Security\_Status parameter is conditional: it is present only if cryptographic protection has been applied. It carries information on the protection that has been verified / removed by the AL. It may be present in all type of service invocations. See Table 58.

The Protection\_Element parameter is conditional: it is present only if the APDU has been authenticated or signed. See Table 58.

**Table 58 – Security parameters**

	<b>.request</b>	<b>.indication</b>	<b>.response</b>	<b>.confirm</b>
Security_Options	C	–	C (=)	–
Security_Options_Element {Security_Options_Element}	M	–	M (=)	–
Security_Protection_Type	M	–	M (=)	–
Glo_Ciphering	S	–	S (=)	–
Ded_Ciphering	S	–	S (=)	–
General_Glo_Ciphering	S	–	S (=)	–
General_Ded_Ciphering	S	–	S (=)	–
General_Ciphering	S	–	S (=)	–
General_Signing	S	–	S (=)	–
<i>With General_Glo_Ciphering and/or General_Ded_Ciphering</i>				
System_Title	U	–	U (=)	–
<i>With General_Ciphering and/or General_Signing</i>				
Transaction_Id	U	–	U (=)	–
Originator_System_Title	U	–	U (=)	–
Recipient_System_Title	U	–	U (=)	–
Date_Time	U	–	U (=)	–
Other_Information	U	–	U (=)	–
<i>With General_Ciphering</i>				
Key_Info_Options	C	–	C (=)	–
Identified_Key_Options	S	–	S (=)	–
Wrapped_Key_Options	S	–	S (=)	–
Agreed_Key_Options	S	–	S (=)	–
<i>With Glo_Ciphering, Ded_Ciphering, General_Glo_Ciphering, General_Ded_Ciphering, General_Ciphering</i>				
Security_Control	M	–	M (=)	–
Security_Status	–	C	–	C (=)
Security_Status_Element {Security_Status_Element}	–	M	–	M (=)
Security_Protection_Type		M		M (=)
Glo_Ciphering	–	S	–	S (=)
Ded_Ciphering	–	S	–	S (=)
General_Glo_Ciphering	–	S	–	S (=)
General_Ded_Ciphering	–	S	–	S (=)
General_Ciphering	–	S	–	S (=)
General_Signing	–	S	–	S (=)
<i>With General_Glo_Ciphering and/or General_Ded_Ciphering</i>				
System_Title	–	U	–	U (=)

**Table 50** (continued)

	.request	.indication	.response	.confirm
<i>With General_Ciphering and/or General_Signing</i>				
Transaction_Id	-	U	-	U (=)
Originator_System_Title	-	U	-	U (=)
Recipient_System_Title	-	U	-	U (=)
Date_Time	-	U	-	U (=)
Other_Information	-	U	-	U (=)
<i>With General_Ciphering</i>				
Key_Info_Status	-	C	-	C (=)
Identified_Key_Status	-	S	-	S (=)
Wrapped_Key_Status	-	S	-	S (=)
Agreed_Key_Status	-	S	-	S (=)
<i>With Glo_Ciphering, Ded_Ciphering, General_Ded_Ciphering, General_Glo_Ciphering, General_Ciphering</i>				
Security_Control	-	M	-	M (=)
<i>The protection element is present when authentication or digital signature is applied.</i>				
Protection_Element {Protection_Element}	-	C	-	C (=)
Invocation_Counter	-	C	-	C (=)
Authentication_Tag	-	C	-	C (=)
Signature	-	C	-	C (=)

The Security\_Options parameter contains one Security\_Options\_Element parameter for each kind of protection to be applied. Similarly, the Security\_Status parameter contains one Security\_Status\_Element parameter for each kind of protection that has been applied. See also 9.2.7.3.

The Security\_Options\_Element and Security\_Status\_Element parameters include the following sub-parameters:

- the Security\_Protection\_Type sub-parameter is mandatory: it identifies the ciphered APDU to be used; see Table 59;
- the System\_Title subparameter is optional. When present, it holds the system title of the sender. It can be present only with General\_Glo\_Ciphering and General\_Ded\_Ciphering.

NOTE 2 The purpose to include system-title of the sender is to allow the other party to build the initialization vector where the system-title has not been exchanged during the media specific registration process or during the AARQ / AARE exchange.

**Table 59 – APDUs used with security protection types**

Security_Protection_Type	APDU
Glo_Ciphering	Service-specific glo-ciphering
Ded_Ciphering	Service-specific ded-ciphering
General_Glo_Ciphering	general-glo-ciphering
General_Ded_Ciphering	general-ded-ciphering
General_Ciphering	general-ciphering
General_Signing	general-signing
See also Table 44.	

The following five parameters are optionally present with General\_Ciphering and General\_Signing:

- Transaction\_Id: identifies the transaction between two parties;
- Originator\_System\_Title: indicates the system title of the originator of the protected APDU;
- Recipient\_System\_Title: indicates the system title of the recipient, i.e. the entity which will verify / remove the protection that has been applied to the APDU. In the case of broadcast, the Recipient\_System\_Title shall be an empty string;
- The Date\_Time parameter is optional. When present, it indicates the date and time of the invocation of the .request / .response service primitive. Unless otherwise specified in a project specific companion specification, the Date\_Time parameter in the response shall be present if it was present in the request and shall not be present in the response if it was not present in the request;
- the Other\_Information parameter is optional. When present, it holds additional information concerning the protection. Its content may be specified in project specific companion specifications;

If any of the parameters above is not used, than an octet-string of length zero shall be included.

The Key\_Info\_Options parameter is conditional: when protection has to be applied, it carries information on the symmetric key that has been used by the originator / is to be used by the recipient. The key information is sent / received as part of the ciphered APDU:

- Identified\_Key\_Options (see 9.2.5.3): it can be used when the partners share the key; this may be the global unicast encryption key or the global broadcast encryption key;
- Wrapped\_Key\_Options (see 9.2.5.4): in this case, a wrapped key is sent;
- Agreed\_Key\_Options (see 9.2.5.5): in this case, the partners use a Diffie-Hellman key agreement scheme to agree on the key;

Security\_Control: contains the Security Control byte, see Table 45.

The Protection\_Element parameter is conditional: it shall be present if the APDU has been authenticated or digitally signed. It may be present in all type of service invocations, but it may be empty if it is not yet available (this may occur in the case when general block transfer is used). It contains:

- in the case of General\_Ciphering, the Invocation\_Counter, holding the invocation field of the initialization vector, see 9.2.3.3.7.3;
- in the case when the APDU has been authenticated, the authentication tag;
- in the case when the APDU has been signed, the digital signature.

### 9.3.6 The GET service

#### 9.3.6.1 Function

The GET service is used with LN referencing. It can be invoked in a confirmed or unconfirmed manner. Its function is to read the value of one or more COSEM object attributes. The result can be delivered in a single response or – if it is too long to fit in a single response – in multiple responses, with block transfer.

#### 9.3.6.2 Semantics of the service primitives

The GET service primitives shall provide parameters as shown in Table 60.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	277/614
-----------------------	------------	-----------------------	---------

**Table 60 – Service parameters of the GET service**

	.request	.indication	.response	.confirm
Invoke_Id	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Service_Class	M	M (=)	M (=)	M (=)
Request_Type	M	M (=)	–	–
COSEM_Attribute_Descriptor { COSEM_Attribute_Descriptor }	C	C (=)	–	–
COSEM_Class_Id	M	M (=)		
COSEM_Object_Instance_Id	M	M (=)		
COSEM_Object_Attribute_Id	M	M (=)		
Access_Selection_Parameters	U	U (=)		
Access_Selector	M	M (=)		
Access_Parameters	M	M (=)		
Block_Number	C	C (=)	–	–
Response_Type	–	–	M	M (=)
Result	–	–	M	M (=)
Get_Data_Result { Get_Data_Result }	–	–	S	S (=)
Data			S	S (=)
Data_Access_Result			S	S (=)
DataBlock_G	–	–	S	S (=)
Last_Block			M	M (=)
Block_Number			M	M (=)
Result			M	M (=)
Raw_Data			S	S (=)
Data_Access_Result			S	S (=)
NOTE For security parameters, see Table 58.				

The Invoke\_Id parameter identifies the instance of the service invocation.

The Priority parameter indicates the priority level associated to the instance of the service invocation: normal (FALSE) or high (TRUE).

The Service\_Class parameter indicates whether the service is confirmed or unconfirmed. The handling of this parameter depends on the communication profile. See 10.

The use of the Request\_Type and Response\_Type parameters is shown in Table 61.

**Table 61 – GET service request and response types**

Request type		Response type	
NORMAL	The value of a single attribute is requested.	NORMAL	The complete result is delivered.
		ONE-BLOCK	One block of the result is delivered.
NEXT	The next data block is requested.	ONE-BLOCK	As above.
		LAST-BLOCK	The last block of the result is delivered.
WITH-LIST	The value of a list of attributes is requested.	WITH-LIST	The complete result is delivered.
		ONE-BLOCK	As above.
NOTE The same Response_Type may be present more than once, to show the possible responses to each kind of request.			

The COSEM\_Attribute\_Descriptor parameter references a COSEM object attribute. It is present when Request\_Type == NORMAL or WITH-LIST. It is a composite parameter:

- the (COSEM\_Class\_Id, COSEM\_Object\_Instance\_Id) doublet non-ambiguously references one and only one COSEM object instance;
- the COSEM\_Object\_Attribute\_Id element identifies the attribute(s) of the object instance. COSEM\_Object\_Attribute\_Id == 0 references all public attributes of the object (Attribute\_0 feature, see 9.1.4.3.7);
- the Access\_Selection\_Parameters is present only when COSEM\_Object\_Attribute\_Id != 0 and if selective access to the given attribute is available; see 9.1.4.3.5. The Access\_Selector and Access\_Parameters sub-parameters are defined in the COSEM object definitions, see DLMS UA 1000-1.

A GET-REQUEST-NORMAL service primitive contains a single COSEM attribute descriptor. A GET-REQUEST-WITH-LIST service primitive contains a list of COSEM attribute descriptors; their number is limited by the server-max-receive-pdu-size: a GET.request service primitive shall always fit in a single APDU.

The Block\_Number parameter is used in the GET-REQUEST-NEXT service primitive. It carries the number of the latest data block received correctly.

If the encoded form of the response fits in a single APDU, the Result is of type Get\_Data\_Result:

- the Data choice carries the value of the attribute at the time of access; or
- the Data\_Access\_Result choice carries the reason for the read to fail for this attribute.

A GET-RESPONSE-NORMAL service primitive carries a single Get\_Data\_Result parameter. A GET-RESPONSE-WITH-LIST service primitive carries a list of Get\_Data\_Result parameters; their number and order shall be the same as that of the COSEM\_Attribute\_Descriptor parameters in the request.

If COSEM\_Object\_Attribute\_Id == 0 (Attribute\_0), the Data shall be a structure containing the value of all public attributes in the order of their appearance in the given object specification. For attributes to which no access right is granted within the given AA, or which cannot be accessed for any other reason, null-data shall be returned.

If the encoded form of the response does not fit in a single APDU, it can be transported in data blocks using either the service-specific or the general block transfer mechanism.

If the service-specific block transfer mechanism is used, the Result is of type DataBlock\_G. It carries block transfer control information and raw-data:

- the Last\_Block element indicates whether the current block is the last one (TRUE) or not (FALSE);
- the Block\_Number element carries the number of the actual block sent;
- the (inner) Result element carries either Raw\_Data or Data\_Access\_Result. Within this:
  - if the value of a single attribute was requested, Raw\_Data carries a part of the value of the attribute (Data);
  - if the Data cannot be delivered, the response should be GET-RESPONSE-NORMAL with Data\_Access\_Result.
  - if the value of a list of attributes was requested, Raw\_Data carries a part of the list of Get\_Data\_Result-s, either Data or Data\_Access\_Result for each attribute;
  - if the raw data cannot be delivered, Data\_Access\_Result shall carry the reason. For error cases, see 9.4.6.3.

### 9.3.6.3 Use

Possible logical sequences of the GET service primitives are illustrated in Figure 112:

- for a successful confirmed GET, item a);
- for an unconfirmed GET item d); and
- for an unsuccessful attempt due to a local error item c).

The GET.request primitive is invoked by the client AP to read the value of one or all attributes of one or more COSEM objects of the server AP. The first request shall be always of Request\_Type == NORMAL or WITH-LIST. A GET-REQUEST-NEXT service primitive is invoked only when the server could not deliver the complete data in a single response, i.e. following the reception of a .confirm primitive of Response\_Type == ONE-BLOCK. Upon the reception of the .request primitive, the client AL builds the Get-Request APDU appropriate for the request type and sends it to the server.

The GET.indication primitive is generated by the server AL upon the reception of a Get-Request APDU.

The GET.response primitive is invoked by the server AP, if Service\_Class == Confirmed, to send a response to an .indication primitive received. If the complete data requested fits in a single APDU, the .response primitive is invoked with Response\_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it is invoked with Response\_Type == ONE-BLOCK and finally with LAST-BLOCK.

The GET.confirm primitive is generated by the client AL to indicate the reception of a Get-Response APDU.

The protocol for the GET service is specified in 9.4.6.3.

### 9.3.7 The SET service

#### 9.3.7.1 Function

The SET service is used with LN referencing. It can be invoked in a confirmed or unconfirmed manner. Its function is to write the value of one or more COSEM object attributes. The data to be written can be sent in a single request or – if it is too long to fit in a single request – in multiple requests, with block transfer.

#### 9.3.7.2 Semantics of the service primitives

The SET service primitives shall provide parameters as shown in Table 62.

**Table 62 – Service parameters of the SET service**

	.request	.indication	.response	.confirm
Invoke_Id	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Service_Class	M	M (=)	M (=)	M (=)
Request_Type	M	M (=)	–	–
COSEM_Attribute_Descriptor { COSEM_Attribute_Descriptor }	C	C (=)	–	–
COSEM_Class_Id	M	M (=)		
COSEM_Object_Instance_Id	M	M (=)		
COSEM_Object_Attribute_Id	M	M (=)		
Access_Selection_Parameters	U	U (=)		
Access_Selector	M	M (=)		
Access_Parameters	M	M (=)		
Data {Data }	C	C (=)	–	–
DataBlock_SA	C	C (=)	–	–
Last_Block	M	M (=)		
Block_Number	M	M (=)		
Raw_Data	M	M (=)		
Response_Type	–	–	M	M (=)
Result { Result }	–	–	C	C (=)
Block_Number	–	–	C	C (=)
NOTE For security parameters, see Table 58.				

The Invoke\_Id parameter identifies the instance of the service invocation.

The Priority parameter indicates the priority level associated to the instance of the service invocation: normal (FALSE) or high (TRUE).

The Service\_Class parameter indicates whether the service is confirmed or unconfirmed. The handling of this parameter depends on the communication profile. See 10.

The use of the Request\_Type and Response\_Type parameters is shown in Table 63.

**Table 63 – SET service request and response types**

<b>Request type</b>		<b>Response type</b>	
NORMAL	The reference of a single attribute and the complete data to be written is sent.	NORMAL	The result is delivered.
FIRST-BLOCK	The reference of a single attribute and the first block of the data to be written is sent.	ACK-BLOCK	The correct reception of the block is acknowledged.
ONE-BLOCK	One block of the data to be written is sent.		
LAST-BLOCK	The last block of the data to be written is sent.	LAST-BLOCK	The correct reception of the last block is acknowledged and the result is delivered.
		LAST-BLOCK-WITH-LIST	The correct reception of the last block is acknowledged and the list of results is delivered.
WITH-LIST	The reference of a list of attributes and the complete data to be written is sent.	WITH-LIST	The list of results is delivered.
FIRST-BLOCK-WITH-LIST	The reference of a list of attributes and the first block of data to be written is sent.	ACK-BLOCK	The correct reception of the block is acknowledged.
NOTE The same Response_Type may be present more than once, to show the possible responses to each request.			

The COSEM\_Attribute\_Descriptor parameter references a COSEM object attribute. It is present when Request\_Type == NORMAL, FIRST-BLOCK, WITH-LIST and FIRST-BLOCK-WITH-LIST. It is a composite parameter:

- the (COSEM\_Class\_Id, COSEM\_Object\_Instance\_Id) doublet non-ambiguously references one and only one COSEM object instance;
- the COSEM\_Object\_Attribute\_Id identifies the attribute(s) of the object instance. COSEM\_Object\_Attribute\_Id == 0 references all public attributes of the object (Attribute\_0 feature, see 9.1.4.3.7);
- the Access\_Selection\_Parameters is present only when COSEM\_Object\_Attribute\_Id != 0 and if selective access to the given attribute is available; see 9.1.4.3.5. The Access\_Selector and the Access\_Parameters sub-parameters are defined in the COSEM object definitions, see DLMS UA 1000-1.

A SET-REQUEST-NORMAL or SET-REQUEST-WITH-FIRST-BLOCK service primitive contains a single COSEM attribute descriptor. A SET-REQUEST-WITH-LIST or a SET-REQUEST-FIRST-BLOCK-WITH-LIST service primitive contains a list of COSEM attribute descriptors; their number is limited by the server-max-receive-pdu-size: all COSEM attribute descriptors – together with a (part of) the data to be written – shall fit in a single APDU.

The Data parameter contains the data necessary to write the value of the attributes referenced. The number and the order of the Data parameters shall be the same as that of the COSEM\_Attribute\_Descriptor parameters.

If COSEM\_Object\_Attribute\_Id == 0 (Attribute\_0), the Data sent shall be a structure, containing, for each public attribute, in the order of their appearance in the given object specification, either a value or null-data, meaning that the given attribute need not be set.

If the encoded form of the Data parameter does not fit in a single APDU, it can be transported in blocks using either the service-specific or the general block transfer mechanism.

If the service-specific block transfer mechanism is used, the DataBlock\_SA parameter carries block transfer control information and raw-data:

- the Last\_Block element indicates whether the current block is the last one (TRUE) or not (FALSE);
- the Block\_Number element carries the number of the actual block sent;
- the Raw\_Data element carries a part of the data to be written.

The Result parameters are present in the .response primitive when Response\_Type != ACK-BLOCK. Their number and order shall be the same as that of the COSEM\_Attribute\_Descriptor parameters in the request. Each Result shall contain either the success to write or a reason for failing to write the attribute referenced (Data\_Access\_Result). When in the .request primitive COSEM\_Object\_Attribute\_Id == 0 (Attribute\_0), the Result shall carry a single result, either success if all attributes were written or a single reason for failure.

The Block\_Number parameter shall be present when Response\_Type == ACK-BLOCK, LAST-BLOCK, or LAST-BLOCK-WITH-LIST. It carries the number of the latest data block received correctly.

### 9.3.7.3 Use

Possible logical sequences of the SET service primitives is illustrated in Figure 112:

- for a successful confirmed SET, item a);
- for an unconfirmed SET, item d); and
- for an unsuccessful attempt due to a local error, item c).

The SET.request primitive is invoked by the client AP to write the value of one or all attributes of one or more COSEM objects of the server AP. If the complete data to be sent fits in a single APDU, the .request primitive shall be invoked with Request\_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it shall be invoked with Request\_Type == FIRST-BLOCK or FIRST-BLOCK-WITH-LIST, then with Request\_Type == ONE-BLOCK and finally with LAST-BLOCK as appropriate. Upon the reception of the .request primitive, the client AL builds the Set-Request APDU appropriate for the Request\_Type and sends it to the server.

The SET.indication primitive is generated by the server AL upon the reception of a Set-Request APDU.

The SET.response primitive is invoked by the server AP, if Service\_Class == Confirmed, to send a response to an .indication primitive received. If the data were sent in a single APDU, the .response primitive is invoked with Response\_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it is invoked with Response\_Type == ACK-BLOCK, and finally with LAST-BLOCK or LAST-BLOCK-WITH-LIST as appropriate.

The SET.confirm primitive is generated by the client AL to indicate the reception of a Set-Response APDU. The protocol for the SET service is specified in 9.4.6.4.

## 9.3.8 The ACTION service

### 9.3.8.1 Function

The ACTION service is used with LN referencing. It can be invoked in a confirmed or unconfirmed manner. Its function is to invoke one or more COSEM objects methods. It comprises two phases:

- in the first phase, the client sends the reference(s) of the method(s) to be invoked, with the method invocation parameters necessary;
- in the second phase, after invoking the methods, the server sends back the result and the return parameters generated by the invocation of the method(s), if any.

If the method invocation parameters are too long to fit in a single request, they are sent in multiple requests (block transfer from the client to the server). If the result and the return parameters are too long to fit in a single response, they are returned in multiple responses (block transfer from the server to the client.)

### 9.3.8.2 Semantics of the service primitives

The ACTION service primitives shall provide parameters as shown in Table 64.

**Table 64 – Service parameters of the ACTION service**

	.request	.indication	.response	.confirm
Invoke_Id	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Service_Class	M	M (=)	M (=)	M (=)
Request_Type	M	M (=)	-	-
COSEM_Method_Descriptor { COSEM_Method_Descriptor }	C	C (=)	-	-
COSEM_Class_Id	M	M (=)		
COSEM_Object_Instance_Id	M	M (=)		
COSEM_Object_Method_Id	M	M (=)		
Method_Invocation_Parameters { Method_Invocation_Parameters }	U	U (=)	-	-
Response_Type	-	-	M	M (=)
Action_Response { Action_Response }	-	-	M	M (=)
Result			M	M (=)
Response_Parameters			U	U (=)
Data			S	S (=)
Data_Access_Result			S	S (=)
DataBlock_SA	C	C (=)	C	C (=)
Last_Block	M	M (=)	M	M (=)
Block_Number	M	M (=)	M	M (=)
Raw_Data	M	M (=)	M	M (=)
Block_Number	C	C (=)	C	C (=)
NOTE For security parameters, see Table 58.				

The Invoke\_Id parameter identifies the instance of the service invocation.

The Priority parameter indicates the priority level associated to the instance of the service invocation: normal (FALSE) or high (TRUE).

The Service\_Class parameter indicates whether the service is confirmed or unconfirmed. The handling of this parameter depends on the communication profile. See 10.

The use of the Request\_Type and Response\_Type parameters is shown in Table 65.

**Table 65 – ACTION service request and response types**

Request type		Response type	
NORMAL	The reference of a single method and the complete method invocation parameter is sent.	NORMAL	The result and the complete return parameter are sent.
		ONE-BLOCK	One block of the result and of the return parameter is sent.
NEXT	The next data block is requested.	ONE-BLOCK	As above.
		LAST-BLOCK	The last block of the result(s) and of the return parameter(s) is sent.
FIRST-BLOCK	The reference of a single method and the first block of the method invocation parameters is sent.	NEXT	The correct reception of the block is acknowledged.
ONE-BLOCK	One block of the method invocation parameters is sent.		
LAST-BLOCK	The last block of the method invocation parameters is sent.	NORMAL	As above.
		ONE-BLOCK	As above.
WITH-LIST	The reference of a list of methods and the complete list of method invocation parameters is sent.	WITH-LIST	The complete list of results and return parameters is sent.
		ONE-BLOCK	See above.
WITH-LIST-AND-FIRST-BLOCK	The reference of a list of methods and the first block of the method invocation parameters is sent.	NEXT	The correct reception of the block is acknowledged.
NOTE The same Response_Type may be present more than once, to show the possible responses to each request.			

The COSEM\_Method\_Descriptor parameter references a COSEM object method. It is present if Request\_Type == NORMAL, FIRST-BLOCK, WITH-LIST and WITH-LIST-AND-FIRST-BLOCK. It is a composite parameter:

- the (COSEM\_Class\_Id, COSEM\_Object\_Instance\_Id) doublet non-ambiguously references one and only one COSEM object instance;
- the COSEM\_Method\_Id identifies one method of the COSEM object referenced.

An ACTION-REQUEST-NORMAL or ACTION-REQUEST-FIRST-BLOCK service primitive shall contain a single COSEM method descriptor. An ACTION-REQUEST-WITH-LIST or ACTION-REQUEST-WITH-LIST-AND-FIRST-BLOCK service primitive shall contain a list of COSEM method descriptors; their number is limited by the server-max-receive-pdu-size: all COSEM method references – together with a (part of) the method invocation parameters – shall fit in a single APDU.

The Method\_Invocation\_Parameter parameter carries the parameter(s) necessary for the invocation of the method(s) referenced.

- if Request\_Type == NORMAL, the Method\_Invocation\_Parameter parameter is optional;
- if Request\_Type == WITH-LIST, the service primitive shall contain a list of Method\_Invocation\_Parameters. The number and the order of the method invocation parameters shall be the same as that of the COSEM\_Method\_Descriptor-s. If the invocation of any of the methods does not require additional parameters, it shall be nevertheless present, but it shall be null data.

If the encoded form of the COSEM method descriptor(s) and method invocation parameter(s) does not fit in a single APDU, it can be transported in blocks using either the service-specific or the general block transfer mechanism.

If the service-specific block transfer mechanism is used the DataBlock\_SA parameter carries block transfer control information and raw-data:

- the Last\_Block element indicates whether the current block is the last one (TRUE) or not (FALSE);
- the Block\_Number element carries the number of the actual block sent;
- the Raw\_Data element carries a part of the method invocation parameters.

The Action\_Response parameters are present in the .response primitive when Response\_Type == NORMAL, or WITH-LIST. Their number and the order shall be the same as that of the COSEM method descriptors. It consists of two elements:

- the Result parameter. It contains either the success to invoke or a reason for failing to invoke the method referenced (Action-Result);
- the Response\_Parameter(s). Each response parameter shall contain either the Data returned as a result of invoking the method, or a reason for returning the parameters to fail (Data-Access-Result). If the invocation of any of the methods does not return parameters, null data shall be returned.

If the response does not fit in a single APDU, it can be transported in blocks using either the service-specific or the general block transfer mechanism.

If the service-specific block transfer mechanism is used, the DataBlock\_SA parameter carries block transfer control information and raw-data:

- the Last\_Block element indicates whether the current block is the last one (TRUE) or not (FALSE);
- the Block\_Number element carries the number of the actual block sent;
- the Raw\_Data element carries a part of the response:
  - if a single method was invoked, Raw\_Data carries the result and the Response\_Parameters if any;
 

NOTE If no Response\_Parameters are returned, the response should be of type ACTION-RESPONSE-NORMAL.
  - if a list of methods was invoked, Raw\_Data carries a part of the list of Action\_Response-s and optional data.

The Block\_Number parameter in an ACTION-REQUEST-NEXT service primitive shall carry the number of the latest data block received from the server correctly.

The Block\_Number parameter in an ACTION-RESPONSE-NEXT service primitive shall carry the number of the latest data block received from the client correctly.

### 9.3.8.3 Use

Possible logical sequences of the ACTION service primitives are illustrated in Figure 112:

- for a successful confirmed ACTION, item a);
- for an unconfirmed ACTION item d); and
- for an unsuccessful attempt due to a local error item c).

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	285/614
-----------------------	------------	-----------------------	---------

In the first phase, the ACTION.request primitive is invoked by the client AP to invoke one or more methods of one or more COSEM objects of the server AP. If the complete list of COSEM method descriptors and method invocation parameters fits in a single APDU, the .request primitive is invoked with Request\_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it is invoked with Request\_Type == FIRST-BLOCK or WITH-LIST-AND-FIRST-BLOCK as appropriate, then with Request\_Type == ONE-BLOCK and finally with LAST-BLOCK. Upon the reception of the .request primitive, the client AL builds the Action-Request APDU appropriate for the Request\_Type and sends it to the server.

The ACTION.indication primitive is generated by the server AL upon the reception of an Action-Request APDU.

During the block transfer of the method invocation parameters, the server AP invokes the ACTION.response primitive with Request\_Type == NEXT until the last block is received.

The ACTION.confirm primitive is generated by the client AL upon the reception of an Action-Response APDU.

Once all method invocation parameters are transferred, the server invokes the methods of COSEM objects referenced, and the second phase commences.

If the complete response fits in a single APDU, the ACTION.response primitive is invoked by the server AP with Response\_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it is invoked with Response\_Type == ONE-BLOCK and finally with LAST-BLOCK. Upon the reception of the .response primitive, the server AL builds the Action-Response APDU appropriate for the Response\_Type and sends it to the client.

The ACTION.confirm primitive is generated by the client AL to indicate the reception of an Action-Response APDU.

During the block transfer of the return parameters, the client AP invokes the .request primitive with Request\_Type == NEXT until the last block is received.

The protocol for the ACTION service is specified in 9.4.6.5.

### 9.3.9 The ACCESS service

#### 9.3.9.1 Overview – Main features

##### 9.3.9.1.1 General

The ACCESS service is a unified service which can be used to access multiple COSEM object attributes and/or methods with a single .request / .response. The purpose of introducing it is to improve xDLMS messaging while maintaining co-existence with the existing xDLMS services.

##### 9.3.9.1.2 Unified WITH-LIST service to improve efficiency

The ACCESS service is a unified service using LN referencing that can be used to read or write multiple COSEM object attributes and/or to invoke multiple methods with a single .request / .response. Each request contains a list of requests and related data. Each response contains a list of return data and the result of the request.

NOTE SN referencing is currently not supported. It can be added by introducing new variants of the service.

Whereas GET- / SET- / ACTION-WITH-LIST service requests can include one request type –GET, SET and/or ACTION – only on the list, ACCESS service requests can include different request types. This allows reducing the number of exchanges and thereby improves efficiency.

The processing of the list of requests starts at the first request on the list and continues with processing the next one until the end is reached.

### 9.3.9.1.3 Specific variants for selective access

The GET / SET .request service primitives shall always contain Access\_Selection\_Parameters even in the case when selective access is not available or not needed. In contrast, the ACCESS service provides specific variants to access attributes without or with selective access. This obviates the need to include Access\_Selection\_Parameters when selective access is not available or not needed thereby reducing overhead and improving efficiency.

### 9.3.9.1.4 Long\_Invoke\_Id parameter

The ACCESS service uses the long\_invoke\_id parameter, see 9.1.4.4.3.

### 9.3.9.1.5 Self-descriptive responses

When requested by the client, the ACCESS.response service primitive carries not only the response to each request, i.e. the result of accessing each attribute / method and the return data but also the Access\_Request\_Specification service parameter – carrying the attribute / method references and where applicable, the Access\_Selection parameters – rendering the .response service primitive self-descriptive. Such self-descriptive responses can be stored and processed on their own, without the need to pair responses and requests.

### 9.3.9.1.6 Failure management

In the case of the GET- / SET- / ACTION-WITH-LIST services the client cannot control what should happen if one of the requests fails. In contrast, the ACCESS service allows the client to control if the requests that follow the failed one on the list should be processed or not.

### 9.3.9.1.7 Time stamp as a service parameter

The xDLMS services specified earlier do not provide a service parameter in the .request or in the .response service primitive to carry a time stamp.

In contrast, ACCESS service primitives provide a service parameter to carry the time stamp holding the date and time of invoking the service primitive. This further reduces overhead.

### 9.3.9.1.8 Presence of data in service primitives

There are important differences between the GET / SET / ACTION services and the ACCESS service as regards to data in the service primitives:

- GET service: data is not present in the request. In the response, either data or result (Data-Access-Result) is returned;
- SET service: data is present in the request. In the response only result (Data-Access-Result) is returned;
- ACTION service: method invocation parameters are optional in the request. In the response the result of invoking the method (Action-Result) and optionally the result of returning the return parameters (Data or Data-Access-Result) is returned;
- ACCESS service: data is associated with each attribute / method reference in the request. If data is not needed for a particular request, then null-data is included. In the response, both data and result are returned. If there is no data to return for a particular response, then null-data is included. In the case of accessing a method, Access-Response-Action (Action-Result) conveys both the result of invoking the method and the result of returning the return parameters.

## 9.3.9.2 Service specification

### 9.3.9.2.1 Function

The ACCESS service is a unified service using LN referencing that can be used to read or write multiple COSEM object attributes and/or to invoke multiple methods with a single .request / .response. Each request contains a list of requests and related data. Each response contains a list of return data and the result of the request. It can be invoked in a confirmed or unconfirmed manner. It can be used with the general block transfer and general ciphering mechanisms.

The use of the conformance block is the following:

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	287/614
-----------------------	------------	-----------------------	---------

- bit 17 *access* indicates the support of the ACCESS service;
- bit 1 *general-protection* indicates the availability of the general protection APDUs;
- bit 2 *general-block-transfer* indicates the availability of the GBT mechanism;
- bit 8 *attribute0-supported-with-set* and bit 10 *attribute0-supported-with-get* (10) are not relevant: attribute0 is always supported;
- bit 9 *priority-mgmt-supported* is relevant;
- bit 14 *multiple-references* is irrelevant: the ACCESS service always supports multiple references;
- bit 21 *selective-access* is relevant. The access selection parameters can be used only if the use of selective access has been successfully negotiated.

### 9.3.9.2.2 Semantics of the service primitives

The ACCESS service primitives shall provide parameters as shown in Table 66.

The Long\_Invoke\_Id, Self\_Descriptive, Processing\_Option, Service\_Class and Priority parameters are mandatory. Their value in the .indication, .response and .confirm service primitives shall be the same as in the .request service primitive. They are carried by the bits of the long-invoke-id-and-priority field of the access-request / access-response APDU:

long-invoke-id (bits 0-23) identifies the instance of the service invocation;

- self-descriptive (bit 28) indicates if the service response shall be not self-descriptive (FALSE) or self-descriptive (TRUE). When set to TRUE, the Access\_Response\_Body parameter shall contain the Access\_Request\_Specification parameter;

NOTE 1 The Access\_Request\_List\_Of\_Data parameter is not included in the .response service primitive.

- processing-option (bit 29) specifies what to do when processing a request on the list fails. When set to FALSE, processing continues. When set to TRUE, processing breaks i.e. the requests on the list that follow the failed one shall not be processed. As described in 9.3.9.1.2, processing of the list of requests shall start at the first request on the list and shall continue with processing the next one until the end of the list is reached;
- service-class (bit 30) indicates whether the service invocation is confirmed (TRUE) or unconfirmed (FALSE);

NOTE 2 The Service\_Class parameter applies to the service invocation, not to the individual requests on the list.

NOTE 3 Depending on the communication profile, the Service\_Class parameter may also determine the frame type to be used to carry the APDU.

- priority (bit 31) indicates the priority level associated to the instance of the service invocation. It may be normal (FALSE) or high (TRUE).

NOTE 4 The Priority parameter applies to the service invocation, not to the individual requests on the list.

The Date\_Time service parameter is optional. When present, it shall contain the date and time of the invocation of the service .request / .response. It is carried by the date-time field – of type OCTET STRING – of the access-request / access-response APDU. When not present, then the OCTET STRING shall be of length 0. Unless otherwise specified in a project specific companion specification, the Date\_Time parameter in the response shall be present if it was present in the request and shall not be present in the response if it was not present in the request.

**Table 66 – Service parameters of the ACCESS service**

	.request	.indication	.response	.confirm
Long_Invoke_Id	M	M (=)	M (=)	M (=)
Self_Descriptive	M	M (=)	M (=)	M (=)
Processing_Option	M	M (=)	M (=)	M (=)
Service_Class	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Date_Time	U	U (=)	U	U (=)
Access_Request_Body	M	M (=)	–	–
Access_Request_Specification	M	M (=)	–	–
{ Access_Request_Specification }				
Access_Request_Get	U	U (=)	–	–
COSEM_Attribute_Descriptor	M	M (=)	–	–
Access_Request_Set	U	U (=)	–	–
COSEM_Attribute_Descriptor	M	M (=)	–	–
Access_Request_Action	U	U (=)	–	–
COSEM_Method_Descriptor	M	M (=)	–	–
Access_Request_Get_With_Selection	U	U (=)	–	–
COSEM_Attribute_Descriptor	M	M (=)	–	–
Access_Selection	M	M (=)	–	–
Access_Selector	M	M (=)	–	–
Access_Parameters	M	M (=)	–	–
Access_Request_Set_With_Selection	U	U (=)	–	–
COSEM_Attribute_Descriptor	M	M (=)	–	–
Access_Selection	M	M (=)	–	–
Access_Selector	M	M (=)	–	–
Access_Parameters	M	M (=)	–	–
Access_Request_List_Of_Data	M	M (=)	–	–
Data { Data }			–	–
Access_Response_Body	–	–	M	M (=)
Access_Request_Specification	–	–	C (=) <sup>1</sup>	C (=)
{ Access_Request_Specification }				
Access_Response_List_Of_Data	–	–	M	M (=)
Data { Data }				
Access_Response_Specification	–	–	M	M (=)
{ Access_Response_Specification }				
Access_Response_Get	–	–	C	C (=)
Result	–	–	M	M (=)
Access_Response_Set	–	–	C	C (=)
Result	–	–	M	M (=)
Access_Response_Action	–	–	C	C (=)
Result	–	–	M	M (=)

<sup>1</sup> When the Access\_Request\_Specification service parameter is present in Access\_Response\_Body, then its value shall be the same as in the .request / .indication primitive.

The Access\_Request\_Body parameter contains the Access\_Request\_Specification and the Access\_Request\_List\_Of\_Data sub-parameters.

The Access\_Request\_Specification parameter carries a list of request specifications. The list may have 0 or more elements. Each request can be any of the following:

Without selective access:

- Access\_Request\_Get carries the COSEM\_Attribute\_Descriptor of an attribute to be read;
- Access\_Request\_Set carries the COSEM\_Attribute\_Descriptor of an attribute to be written;
- Access\_Request\_Action carries the COSEM\_Method\_Descriptor of a method to be invoked.

With selective access:

- Access\_Request\_Get\_With\_Selection carries the COSEM\_Attribute\_Descriptor of an attribute to be read with selective access, and the Access\_Selection parameter that contains Access\_Selector and Access\_Parameters;
- Access\_Request\_Set\_With\_Selection carries the COSEM\_Attribute\_Descriptor of an attribute to be written with selective access, and the Access\_Selection parameter that contains Access\_Selector and Access\_Parameters.

Access\_Request\_Get\_ / Set\_ With\_Selection should not be used if selective access to the attribute is not available or if COSEM\_Attribute\_Descriptor identifies all attributes (Attribute\_0).

The COSEM\_Attribute\_Descriptor parameter is a composite parameter:

- the (COSEM\_Class\_Id, COSEM\_Object\_Instance\_Id) doublet non-ambiguously references one and only one COSEM object instance;
- the COSEM\_Object\_Attribute\_Id element identifies the attribute of the object instance. COSEM\_Object\_Attribute\_Id == 0 references all public attributes of the object.

The Access\_Selection parameter carries the Access\_Selector and the Access\_Parameters sub-parameters. The possible values are defined in the relevant COSEM interface class definitions.

The Access\_Request\_List\_Of\_Data parameter carries the list of data related to the list of Access\_Request\_Specification parameters. The data depend on the kind of access request:

- Access\_Request\_Get referencing one or all attributes(Attribute\_0): null-data;
- Access\_Request\_Set: the corresponding data carries the value to be written. In the case of referencing all attributes (Attribute\_0), the data shall be a structure. The number of elements in the structure shall be the same as the number of attributes specified in the relevant COSEM IC specification. Each element in the structure contains the value to be written or null-data meaning that the given attribute need not be written;
- Access\_Request\_Action: the corresponding data carries the method invocation parameters, or if not required null-data.

The number and order of the elements on the two lists shall be the same.

The Access\_Response\_Body parameter contains the following sub-parameters:

- the Access\_Request\_Specification (optionally, only when the Self\_Descriptive == TRUE);
- the Access\_Response\_List\_Of\_Data; and
- the Access\_Response\_Specification.

Notice that in the response Access\_Request\_List\_Of\_Data comes first followed by Access\_Response\_Specification. If data is provided but the result indicates a failure, then the client should discard the data.

The Access\_Request\_Specification parameter, when present, shall be the same as in the .request / .indication primitives.

The Access\_Response\_List\_Of\_Data parameter carries the data resulting from processing the requests. The number of elements on the list shall be the same as on the Access\_Request\_Specification list. The data depend on the kind of access request:

- Access\_Request\_Get referencing a single attribute: the value of the attribute requested or null-data when the value of the attribute cannot be returned;
- Access\_Request\_Get referencing all attributes (Attribute\_0): data shall be a structure, containing the value of each attribute. The number of elements in the structure shall be the same as the number of attributes specified in the relevant COSEM IC specification. If the value of an attribute cannot be returned, then null-data shall be included for that attribute. If no attribute values can be returned then a single null-data shall be returned;
- Access\_Request\_Set referencing one or all attributes (Attribute\_0): null-data;
- Access\_Request\_Action: the return parameters or when not provided, null-data.

The Access\_Response\_Specification parameter carries the result of each request. The number of elements on this list shall be the same as on the Access\_Request\_Specification list:

- Access\_Request\_Get referencing a single attribute: the result of reading the attribute: *success* or reason for the failure;
- Access\_Request\_Get referencing all attributes (Attribute\_0): the result shall be *success* if the value of all attributes to which access right is granted could be returned. Otherwise, it shall be *Data-Access-Result* giving a reason for the failure;
- Access\_Request\_Set referencing a single attribute: the result of writing the attribute: *success* or a reason for the failure;
- Access\_Request\_Set referencing all attributes (Attribute\_0): the result shall be *success* if the value of all attributes to which access right is granted could be written. Otherwise, it shall be *Data-Access-Result* giving a reason for the failure;
- Access\_Response\_Action carries the result of invoking a method: *success* or a reason for the failure. The result shall be *success* if the method could be invoked successfully and – when the IC specification specifies return parameters – they could be successfully returned. Otherwise, it shall be *Action-Result* giving a reason for the failure.

If the Processing\_Option parameter is set to TRUE and processing a request on the list fails, then for all requests following the failed one, Access\_Response\_List\_Of\_Data shall contain null-data and Access\_Response\_Specification shall carry the reason for the failure.

#### 9.3.9.2.3 Use

Possible logical sequences of the ACCESS service primitives are illustrated in Figure 112:

- for a successful confirmed ACCESS, item a);
- for an unconfirmed ACCESS item d); and
- for an unsuccessful attempt due to a local error item c).

The ACCESS.request primitive is invoked by the client AP to read or write the value of a list of COSEM object attributes and/or to invoke a list of methods.

The ACCESS.indication primitive is generated by the server AL upon the reception of an Access-Request APDU.

The ACCESS.response primitive is invoked by the server AP – if Service\_Class == Confirmed – to send a response to an .indication primitive received.

The ACCESS.confirm primitive is generated by the client AL to indicate the reception of an access-response APDU.

When the request or the response does not fit in a single APDU, then the general block transfer mechanism can be used. See 9.1.4.4.9.

If the response would be too long to fit in a single APDU but GBT is not supported, the response may be either a list of null-data and a list of results indicating the reason for the failure.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	291/614
-----------------------	------------	-----------------------	---------

When cryptographic protection is required, the access-request / access-response APDUs can be transported in general-ded-ciphering, general-glo-ciphering, general-ciphering or general-signing APDUs depending on the kind of protection to be applied. See 9.3.5

The protocol of the ACCESS service is specified in 9.4.6.6.

### 9.3.10 The DataNotification service

#### 9.3.10.1 Function

The DataNotification service is an unsolicited, unconfirmed or confirmed service. It is used by the server to push data to the client. The push process is configured by Push setup objects; see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.8.

#### 9.3.10.2 Semantics of the service primitives

The DataNotification service primitives shall provide parameters as shown in Table 67.

**Table 67 – Service parameters of the DataNotification service primitives**

	.request	.indication	.response	.confirm
Long_Invoke_Id	M	M (=)	M (=)	M (=)
Self_Descriptive	-	-	-	-
Processing_Option	-	-	-	-
Service_Class	M	M (=)	M (=)	M (=)
Priority	M	M (=)	-	-
Date_Time	U	U (=)	U	U (=)
Notification_Body	M	M (=)	-	-
Result	-	-	-	M

The Long\_Invoke\_Id parameter identifies the instance of the service invocation. See also 9.1.4.4.3.

The Self\_Descriptive, Processing\_Option and Service\_Class parameters are not used in the case of this service.

The Service\_Class parameter indicates whether the DataNotification.request service primitive is invoked in a confirmed or unconfirmed manner.

The Priority parameter indicates the priority level associated to the instance of the service invocation: normal (FALSE) or high (TRUE).

The optional Date\_Time parameter indicates the time at which the DataNotification.request / DataNotification.response service primitive is invoked. When the DataNotification.confirm primitive is invoked as a result of a lower layer confirmation, no Date\_Time is provided.

The Notification\_Body parameter contains the push data.

The Result parameter indicates the confirmation result and is CONFIRMED in case of Service\_Class == Confirmed and received Data-Notification-Confirm APDU; or in case of Service\_Class == Unconfirmed and received supporting protocol layer confirmation. The Result parameter is SUPPORTING\_LAYER\_FAILED in case of supporting protocol layer failure.

If the encoded form of the request does not fit in a single APDU, it can be transported in data blocks using the general block transfer mechanism.

#### 9.3.10.3 Use

Possible logical sequences of the DataNotification service primitives are illustrated in Figure 112 a), b) and d).

The .request primitive is invoked by the server AP to push data to the remote client AP. Upon the reception of the .request primitive, the server AL builds the DataNotification APDU.

The DataNotification.indication primitive is generated by the client AL upon the reception of a DataNotification APDU.

The DataNotification.response primitive is invoked by the client AP to confirm the receipt of a confirmable DataNotification. Upon the reception of the .response primitive, the client AL builds the Data-Notification-Confirm APDU and sends it to the server.

The DataNotification.confirm primitive is invoked by the server AL to convey the receipt of a Data-Notification-Confirm APDU or lower layer confirmation to the server AP.

The protocol for the DataNotification service is specified in 9.4.6.7.

### 9.3.11 The EventNotification service

#### 9.3.11.1 Function

The EventNotification service is an unsolicited service initiated by the server upon the occurrence of an event in order to inform the client of the value of an attribute as though it had been requested by the COSEM. It is an unconfirmed service.

#### 9.3.11.2 Semantics of the service primitives

The EventNotification service primitives shall provide parameters as shown in Table 68.

**Table 68 – Service parameters of the EventNotification service primitives**

	.request	.indication
Time	U	U (=)
Application_Addresses	U	U (=)
COSEM_Attribute_Descriptor	M	M (=)
COSEM_Class_Id	M	M (=)
COSEM_Object_Instance_Id	M	M (=)
COSEM_Object_Attribute_Id	M	M (=)
Attribute_Value	M	M (=)

The optional Time parameter indicates the time at which the EventNotification.request service primitive was issued.

The Application\_Addresses parameter is optional. It is present only when the EventNotification service is invoked outside of an established AA. In this case, it contains all protocol specific parameters required to identify the sender and destination APs.

When the .request primitive does not contain the optional Application\_Addresses parameter, default addresses shall be used, that of the server Management Logical Device and the Client Management AP. Both APs are always present and in any protocol profile, they are bound to known, pre-defined addresses.

The (COSEM\_Class\_Id, COSEM\_Object\_Instance\_Id, COSEM\_Object\_Attribute\_Id) triplet references non-ambiguously one and only one attribute of a COSEM object instance.

The Attribute\_Value parameter carries the value of this attribute. More information about the notified event may be obtained by interrogating this COSEM object.

If the encoded form of the request does not fit in a single APDU, it can be transported in data blocks using the general block transfer mechanism.

#### 9.3.11.3 Use

A possible logical sequence of the EventNotification service primitives is illustrated in Figure 112 f) and g).

The .request primitive is invoked by the server AP to send the value of a COSEM object attribute to the remote client AP. Upon the reception of the .request primitive, the server AL builds the event-notification-request APDU.

In some cases, the supporting layer protocol(s) do (does) not allow sending a protocol data unit in a real, unsolicited manner. In these cases, the client has to explicitly solicit sending an EventNotification frame, by invoking the TriggerEventNotificationSending service primitive.

The EventNotification.indication primitive is generated by the client AL upon the reception of an event-notification-request APDU.

The protocol for the EventNotification service is specified in 9.4.6.8.

### 9.3.12 The TriggerEventNotificationSending service

#### 9.3.12.1 Function

The function of the TriggerEventNotificationSending service is to trigger the server by the client to send the frame carrying the EventNotification.request APDU.

This service is necessary in the case of protocols, when the server is not able to send a real non-solicited EventNotification.request APDU.

#### 9.3.12.2 Semantics of the service primitive

The TriggerEventNotificationSending.request service primitive shall provide parameters as shown in Table 69.

**Table 69 – Service parameters of the TriggerEventNotificationSending.request service primitive**

	.request
Protocol_Parameters	M

The Protocol\_Parameters parameter contains all lower protocol layer dependent information, which is required for triggering the server to send out an eventually pending frame containing an EventNotification.request APDU. This information includes the protocol identifier, and all the required lower layer parameters.

#### 9.3.12.3 Use

Upon the reception of a TriggerEventNotificationSending.request service invocation from the client AP, the client AL shall invoke the corresponding supporting protocol layer service to send a trigger message to the server.

### 9.3.13 Variable access specification

Variable\_Access\_Specification is a parameter of the xDLMS Read / Write / UnconfirmedWrite InformationReport .request / .indication service primitives. Its variants are shown in Table 70:

- Variable\_Name identifies a DLMS named variable;
- Parameterized\_Access provides the capability to transport additional parameters;
- Block\_Number\_Access transports a block number;
- Read\_Data\_Block\_Access transports block transfer control information and raw data;
- Write\_Data\_Block\_Access transports block transfer control information.

The use of the different variants depends on the service and it is described in the respective SN service specifications.

**Table 70 – Variable Access Specification**

Variable_Access_Specification	Read .request	Write .request	Unconfirmed Write.request	Information Report
Kind_Of_Access	M	M	M	M
Variable_Name	S	S	S	M
Detailed_Access	Not used in DLMS/COSEM			
Parameterized_Access	S	S	S	-
Variable_Name	M	M	M	
Selector	U	U	U	
Parameter	U	U	U	
Block_Number_Access	S	-	-	-
Block_Number	M			
Read_Data_Block_Access	S	-	-	-
Last_Block	M			
Block_Number	M			
Raw_Data	M			
Write_Data_Block_Access	-	S	-	-
Last_Block		M		
Block_Number		M		

### 9.3.14 The Read service

#### 9.3.14.1 Function

The Read service is used with SN referencing. It is a confirmed service. Its functions are:

- to read the value of one or more COSEM object attributes. In this case, the encoded form of the .request service primitive shall fit in a single APDU. The result can be delivered in a single response, or – if it is too long to fit in a single response – in multiple responses, with block transfer;
- to invoke one or more COSEM object methods, when return parameters are expected. In this case, if either the .request (including the method references and the method invocation parameters) or the .response service primitive (including the results and return parameters) is too long to fit in a single APDU, then block transfer with multiple requests and/or responses can be used.

The Read service is specified in IEC 61334-4-41:1996, 10.4 and Annex A. For completeness and for consistency with the specification of services using LN referencing, the specification is reproduced here, together with the extensions made for DLMS/COSEM.

#### 9.3.14.2 Semantics of the service primitives

The Read service primitives shall provide service parameters as shown in Table 71.

**Table 71 – Service parameters of the Read service**

	.request	.indication	.response	.confirm
Variable_Access_Specification { Variable_Access_Specification }	M	M (=)		
Variable_Name	S	S (=)		
Parameterized_Access	S	S (=)	-	-
Variable_Name	M	M (=)		
Selector	U	U (=)		
Parameter	U	U (=)		
Read_Data_Block_Access	S	S (=)		
Last_Block	M	M (=)		
Block_Number	M	M (=)		
Raw_Data	M	M (=)		

	.request	.indication	.response	.confirm
Block_Number_Access Block_Number	S M	S (=) M (=)		
Result (+)			S	S (=)
Read_Result { Read_Result }	-	-	M	M (=)
Data Data_Access_Error			S S	S (=) S (=)
Data_Block_Result Last_Block Block_Number Raw_Data Block_Number			S M M M S	S (=) M (=) M (=) M (=) S (=)
Result (-) Error_Type	-	-	S M	S (=) M (=)
NOTE For security parameters, see 9.3.5.				

The use of the different variants of the Variable-Access-Specification service parameter of the Read.request service primitive and the different choices of the Read.response primitive are shown in Table 72.

If the encoded form of the response does not fit in a single APDU, it can be transported in data blocks using the general block transfer mechanism.

**Table 72 – Use of the Variable\_Access\_Specification variants and the Read.response choices**

Read.request Variable_Access_Specification		Read.response CHOICE	
Variable_Name (Variable_Name)	References a list <sup>1</sup> of COSEM object attributes.	Data {Data}	Delivers the value of the attribute(s) referenced.
		Data_Access_Error {Data_Access_Error}	Provides the reason for the read to fail.
		Data_Block_Result	Delivers block transfer control information and one block of raw data.
Parameterized_Access {Parameterized_Access}	References a list <sup>1</sup> of COSEM object attributes to be read selectively.	Data {Data}	As above.
		Data_Access_Error {Data_Access_Error}	
		Data_Block_Result	
Read_Data_Block_Access	References a list <sup>1</sup> of COSEM object methods, with method invocation parameters.	Data {Data}	Delivers the method invocation return parameters. NOTE If parameters are returned, this implies that the method invocation succeeded.
		Data_Access_Error {Data_Access_Error}	Provides the reason for the method invocation to fail.
		Data_Block_Result	As above.
Block_Number_Access	Carries block transfer control information and one part of encoded form of the COSEM method references and method invocation parameters.	Block_Number	Carries the number of the latest data block received.

Read.request Variable_Access_Specification	Read.response CHOICE
NOTE The same Read.response choice may be present more than once, to show the possible responses to each request.	
<sup>1</sup> A list may have one or more elements.	

The Read.request service primitive may have one or more Variable\_Access\_Specification parameters.

- the Variable\_Name variant is used to reference a complete COSEM object attribute to be read. The request may include one or more variable names;
- the Parameterized\_Access variant is used either:
  - to reference a COSEM object attribute to be read selectively. In this case, the Variable\_Name element references the COSEM object attribute, the Selector and the Parameter elements carry the access selector and the access parameters respectively as specified in the attribute specification; or
  - to reference a COSEM object method to be invoked. In this case, the Variable\_Name element references the method, the Selector element is zero and the Parameter element carries the method invocation parameters (if any) or null data;
  - the request may include one or more parameterized access parameters;

NOTE 1 With this, the Read service can transport information in both directions, just like the ACTION service used with LN referencing: method invocation parameters from the client to the server and return parameters from the server to the client.

- the Read\_Data\_Block\_Access variant is used when one or more COSEM object methods are invoked and the encoded form of the request does not fit in a single APDU. The request may include a single Read\_Data\_Block\_Access parameter. It carries block transfer control information and raw data:
  - the Last\_Block element indicates if the given block is the last one (TRUE) or not (FALSE);
  - the Block\_Number element carries the number of the actual block sent;
  - the Raw\_Data element carries a part of the encoded form of the list of Variable\_Access\_Specification parameters (as it would be used without block transfer) including the method references and the method invocation parameters. Here, only the variants Variable\_Name and Parameterized\_Access are allowed;
- the Block\_Number\_Access variant is used when the server uses block transfer to send a long response, to confirm the reception of a data block and to request the next data block. The request may include a single Block\_Number\_Access parameter. It carries the number of the latest data block received correctly.

The Result (+) parameter indicates that the requested service has succeeded.

Without block transfer, the .response / .confirm service primitives contain one or more Read\_Result parameters. Their number and order shall be the same as that of the Variable\_Name / Parameterized\_Access parameters in the .request / .indication primitives.

If the Read service is used to read attribute(s), then:

- the Data choice is taken to carry the value of the attribute at the time of access;
- the Data\_Access\_Error is taken to carry the reason for the read to fail for this attribute.

If the Read service is used to invoke method(s), then:

- the Data choice is taken to carry the return parameters (if data is returned, this implies that the method invocation succeeded). If there are no return parameters, Data should be null data.  
NOTE 2 However, if no return data is expected, the Write service should be used to invoke methods.
- the Data\_Access\_Error choice is taken to carry the reason for the method invocation to fail for this method.

In the case of block transfer, the .response / .confirm primitive contains a single Read\_Result parameter. The Data\_Block\_Result choice is taken to carry one block of the response:

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	297/614
-----------------------	------------	-----------------------	---------

- the Last\_Block element indicates whether the given block is the last one (TRUE) or not (FALSE);
- the Block\_Number element carries the number of the block sent;
- the Raw\_Data element contains a part of the encoded form of the list of Read\_Results.

If the data block cannot be provided, then the .response primitive shall carry a single Result parameter using the Data\_Access\_Error choice, carrying an appropriate error message: for example (14) data-block-unavailable.

If the block number in the request is not the one expected, or if the next block cannot be delivered, then the Read.response service primitive shall be returned with a single Read\_Result parameter, with the choice Data\_Access\_Error, carrying an appropriate code, for example (19) data-block-number-invalid.

The Block\_Number choice is taken when the Read service is used to invoke one or more methods and the request is sent in several blocks, to confirm the correct reception of a data block and to ask for the next block. It carries the number of the latest block received.

The Result (-) parameter indicates that the service previously requested failed. The Error\_Type parameter provides the reason for failure. In this case, the server shall send back a confirmedServiceError APDU instead of a ReadResponse APDU.

#### 9.3.14.3 Use

A possible logical sequence of the Read service primitives is illustrated in Figure 112 item a).

The Read.request primitive is invoked following the invocation of a GET or ACTION .request primitive by the client AP and mapping this to a Read.request primitive by the Client SN\_Mapper ASE. The client AL builds then the readRequest APDU and sends it to the server. For LN / SN service mapping, see 9.4.6.9.

The Read.indication primitive is generated by the server AL upon the reception of a readRequest APDU.

The Read.response primitive is invoked by the server AP in order to send a response to a previously received Read.indication primitive. The server AL builds then the readResponse APDU and sends it to the client.

The Read.confirm primitive is generated by the client AL following the reception of a readResponse APDU. It is mapped then back to a GET or ACTION .confirm primitive by the Client SN\_Mapper ASE and the GET or ACTION .confirm primitive is generated.

The protocol of the Read service is specified in 9.4.6.9.

#### 9.3.15 The Write service

##### 9.3.15.1 Function

The Write service is used with SN referencing. It is a confirmed service. Its functions are:

- to write the value of one or more COSEM object attributes;
- to invoke one or more COSEM object methods when no return parameters are expected.

In both cases, if the encoded form of the .request service primitive does not fit in a single APDU, then it can be sent in several requests with block transfer. The .response service primitive shall always fit in a single APDU.

The Write service is specified in IEC 61334-4-41:1996, 10.5 and Annex A. For completeness and for consistency with the specification of services using LN referencing, the specification is reproduced here, together with the extensions made for DLMS/COSEM.

##### 9.3.15.2 Semantics of the service primitives

The Write service primitives shall provide service parameters as shown in Table 73.

**Table 73 – Service parameters of the Write service**

	.request	.indication	.response	.confirm
Variable_Access_Specification { Variable_Access_Specification }	M	M(=)		
Variable_Name	S	S (=)		
Parameterized_Access	S	S (=)	-	-
Variable_Name	M	M (=)		
Selector	M	M (=)		
Parameter	M	M (=)		
Write_Data_Block_Access	S	S (=)		
Last_Block	M	M (=)	-	-
Block_Number	M	M (=)		
Data { Data }	M	M (=)	-	-
Result (+)	-	-	S	S (=)
Write_Result { Write_Result }	-	-	S	S (=)
Success			S	S (=)
Data_Access_Error			S	S (=)
Block_Number			S	S (=)
Result (-)			S	S (=)
Error_Type			M	M (=)
NOTE	For security parameters, see Table 58.			

The use of the different variants of the Variable-Access-Specification service parameter of the Write.request service primitive and the different choices of the Write.response primitive are shown in Table 74. The use of the Data service parameter is also explained.

If the encoded form of the request does not fit in a single APDU, it can be transported in data blocks using either the service-specific or the general block transfer mechanism.

**Table 74 – Use of the Variable\_Access\_Specification variants and the Write.response choices**

Write.request Variable_Access_Specification		Write.response CHOICE	
Variable_Name {Variable_Name}	References a list <sup>1</sup> of COSEM object attributes. The Data service parameter carries the data to be written or the method invocation parameter(s).	Success {Success}	Indicates that the attribute referenced could be successfully written.
		Data_Access_Error {Data_Access_Error}	Provides the reason for the write to fail.
Parameterized_Access {Parameterized_Access}	References a list <sup>1</sup> of COSEM object attributes to be written selectively. The Data service parameter carries the data to be written.	Success {Success}	As above.
		Data_Access_Error {Data_Access_Error}	
Write_Data_Block_Access	Carries block transfer control information. The Data service parameter carries raw-data, including the encoded form of the list <sup>1</sup> of COSEM object attribute or method references, and the list of data to be written or the list of method invocation parameters.	Block_Number	Carries the number of the latest data block received.
NOTE The same Write.response choice may be present more than once, to show the possible responses to each request.			
<sup>1</sup> A list may have one or more elements.			

The Write.request service primitive may have one or more Variable\_Access\_Specification parameters:

- the Variable\_Name variant is used to reference a complete COSEM object attribute to be written or COSEM object method to be invoked. The request may include one or more variable names;
- the Parameterized\_Access variant is used to reference a COSEM object attribute to be written selectively. In this case, the Variable\_Name element references the COSEM object attribute, the Selector and the Parameter elements carry the access selector and the access parameters respectively as specified in the attribute specification. The request may include one or more Parameterized\_Access parameters.

The Data service parameter carries the value(s) to be written to the attribute(s), or the method invocation parameter(s) of the method(s) to be invoked. The number and the order of the Data parameters shall be the same as that of the Variable\_Access\_Specification parameters.

If the Write.request service primitive does not fit into a single APDU, block transfer may be used. In this case:

- the Write\_Data\_Block\_Access variant of the Variable\_Access\_Specification carries block transfer control information:
  - the Last\_Block element indicates whether the given block is the last one (TRUE) or not (FALSE);
  - the Block\_Number element carries the number of the actual block sent;
- the Data parameter carries one part of the list of the attribute references and the list of data to be written, or one part of the list of method references and the list of method invocation parameters;
- The request includes a single Write\_Data\_Block\_Access and a single Data parameter.

The Result (+) parameter indicates that the service requested has succeeded.

The .response / .confirm service primitives contain a list of Write\_Result parameters. Their number and order shall be the same as that of the Variable\_Name / Parameterized\_Access parameters in the .request / .indication service primitives.

Without block transfer, and with block transfer after receiving the last block:

- when the Write service is used to write attribute(s), each element carries either the success of the write access (Success) or a reason for the write to fail for this variable (Data\_Access\_Error);
- when the Write service is used to invoke method(s), each element carries either the success of the method invocation access (Success) or a reason for the method invocation to fail for this variable (Data\_Access\_Error).

The Block\_Number choice is used during block transfer to confirm the correct reception of a data block and to ask for the next block. It carries the number of the latest block received.

If the block-number in the request is not the one expected, or if the block could not be received correctly, then the Write.response service primitive shall be returned with a single Write\_Result parameter, with the choice Data\_Access\_Error, carrying an appropriate code, for example (19) data-block-number-invalid.

The Result (-) parameter indicates that the service requested has failed. The Error\_Type parameter provides the reason for failure. In this case, the server shall send back a confirmedServiceError APDU instead of a writeResponse APDU.

### 9.3.15.3 Use

A possible logical sequence of the Write service primitives is illustrated in Figure 112 item a).

The Write.request primitive is invoked following the invocation of a SET or ACTION .request primitive by the client AP and mapping this to a Write.request primitive by the Client SN\_Mapper ASE. The client AL builds then the writeRequest APDU and sends it to the server. For LN / SN service mapping, see 9.4.6.10.

The Write.indication primitive is generated by the server AL upon the reception of a WriteRequest APDU.

The Write.response primitive is invoked by the server AP in order to send a response to a previously received Write.indication primitive. The server AL builds then the writeResponse APDU and sends it to the client.

The Write.confirm primitive is generated by the client AL following the reception of a writeResponse APDU. It is mapped then back to a SET or ACTION .confirm primitive by the Client SN\_Mapper ASE and the SET or ACTION .confirm primitive is generated.

The protocol of the Write service is specified in 9.4.6.10.

### 9.3.16 The UnconfirmedWrite service

#### 9.3.16.1 Function

The UnconfirmedWrite service is used with SN referencing. It is an unconfirmed service. Its functions are:

- to write the value of one or more COSEM object attributes;
- to invoke one or more COSEM object method when no return parameters are expected.

The UnconfirmedWrite.request service primitive shall always fit in a single APDU.

The UnconfirmedWrite service is specified in IEC 61334-4-41:1996, 10.6 and Annex A. For completeness and for consistency with the specification of services using LN referencing, the specification is reproduced here, together with the extensions made for DLMS/COSEM.

#### 9.3.16.2 Semantics of the service primitives

The UnconfirmedWrite service primitives shall provide service parameters as shown in Table 75.

**Table 75 – Service parameters of the UnconfirmedWrite service**

	.request	.indication
Variable_Access_Specification { Variable_Access_Specification }	M	M(=)
Variable_Name	S	S (=)
Parameterized_Access	S	S (=)
Variable_Name	M	M (=)
Selector	M	M (=)
Parameter	M	M (=)
Data { Data }	M	M (=)
NOTE For security parameters, see Table 58.		

The use of the different variants of the Variable-Access-Specification service parameter of the UnconfirmedWrite.request service primitive is shown in Table 76. The use of the Data service parameter is also explained.

If the encoded form of the request does not fit in a single APDU, it can be transported in data blocks using the general block transfer mechanism.

**Table 76 – Use of the Variable\_Access\_Specification variants**

UnconfirmedWrite.request Variable_Access_Specification	
Variable_Name {Variable_Name}	References a COSEM object attribute. The Data service parameter carries the data to be written or the method invocation parameter(s).
Parameterized_Access {Parameterized_Access}	References a COSEM object attribute with selective access.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	301/614
-----------------------	------------	-----------------------	---------

<b>UnconfirmedWrite.request Variable_Access_Specification</b>	
	The Data service parameter carries the data to be written.

The UnconfirmedWrite.request service primitive may have one or more Variable\_Access\_Specification parameters:

- the Variable\_Name variant is used to reference a complete COSEM object attribute to be written or COSEM object method to be invoked;
- the Parameterized\_Access variant is used to reference a COSEM object attribute to be written selectively. In this case, the Variable\_Name element references the COSEM object attribute, the Selector and the Parameter elements carry the access selector and the access parameters respectively as specified in the attribute specification.

The Data service parameter carries the value(s) to be written to the attribute(s), or the method invocation parameter(s) of the method(s) to be invoked. The number and the order of the Data parameters shall be the same as that of the Variable\_Access\_Specification parameters.

### 9.3.16.3 Use

A possible logical sequence of the Write service primitives is illustrated in Figure 112 item d).

The UnconfirmedWrite.request primitive is invoked following the invocation of a SET or ACTION .request primitive with Service\_Class == Unconfirmed by the client AP and mapping this to an UnconfirmedWrite.request primitive by the Client SN\_Mapper ASE. The client AL builds then the unconfirmedWriteRequest APDU and sends it to the server.

The UnconfirmedWrite.indication primitive is generated by the server AL upon the reception of a unconfirmedWriteRequest APDU.

The protocol of the UnconfirmedWrite service is specified in 9.4.6.11.

## 9.3.17 The InformationReport service

### 9.3.17.1 Function

The InformationReport service is an unsolicited service initiated by the server upon the occurrence of an event in order to inform the client of the value of one or more DLMS named variables – mapped to COSEM object attributes – as though they had been requested by the client. It is an unconfirmed service.

The InformationReport service is specified in IEC 61334-4-41:1996, 10.7 and Annex A. For completeness and for consistency with the specification of services using LN referencing, the specification of the InformationReport service is reproduced here, together with the extensions made for DLMS/COSEM.

### 9.3.17.2 Semantics of the service primitives

The InformationReport service primitives shall provide parameters as shown in Table 77.

**Table 77 – Service parameters of the InformationReport service**

	<b>.request</b>	<b>.indication</b>
Current_Time	M	M (=)
Variable_Access_Specification { Variable_Access_Specification }	M	M (=)
Variable_Name	M	M (=)
Data { Data }	M	M(=)

The Current\_Time parameter indicates the time at which the InformationReport.request service primitive was issued.

The Variable\_Access\_Specification parameter of choice Variable\_Name specifies one or more DLMS named variables – mapped to COSEM object attributes – the value of which is sent by the server.

The Data parameter carries the value of the DLMS named variable(s), in the same order as the order of the Variable\_Access\_Specification parameter(s).

The protocol for the InformationReport service is specified in 9.4.6.12.

### 9.3.18 Client side layer management services: the SetMapperTable.request

#### 9.3.18.1 Function

The function of the SetMapperTable service is to manage the Client SN\_Mapper ASE.

#### 9.3.18.2 Semantics of the service primitive

There is only one primitive, the .request primitive. It shall provide parameters as follows, see Table 78.

**Table 78 – Service parameters of the SetMapperTable.request service primitives**

	.request
Mapping_Table	M

The Mapping\_table parameter is mandatory. It contains the contents of the attribute *object\_list* for the requested server and AA. The structure of the content is defined in DLMS UA 1000-1.

#### 9.3.18.3 Use

The SetMapperTable.request service is invoked by the client AP to provide mapping information to the Client SN\_Mapper ASE. This service does not cause any data transmission between the client and the server. The client AP uses this service primitive, in order to enhance the efficiency of the mapping process if SN referencing is used.

### 9.3.19 Summary of services and LN/SN data transfer service mapping

Table 79 and Table 80 provide a summary of the DLMS/COSEM application layer services.

**Table 79 – Summary of ACSE services**

Client side	Server side
COSEM-OPEN.request	COSEM-OPEN.indication
COSEM-OPEN.confirm	COSEM-OPEN.response
COSEM-RELEASE.request	COSEM-RELEASE.indication
COSEM-RELEASE.confirm	COSEM-RELEASE.response
COSEM-ABORT.indication	COSEM-ABORT.indication

**Table 80 – Summary of xDLMS services**

Client side	Server side
<b>LN referencing</b>	
GET.request	GET.indication
GET.confirm	GET.response
SET.request	SET.indication
SET.confirm	SET.response
ACTION.request	ACTION.indication
ACTION.confirm	ACTION.response

ACCESS.request	ACCESS.indication
ACCESS.confirm	ACCESS.response
EventNotification.indication	EventNotification.request
TriggerEventNotificationSending.request	–
DataNotification.indication	DataNotification.request
DataNotification.response	DataNotification.confirm
<b>SN referencing</b>	
Read.request	Read.indication
Read.confirm	Read.response
Write.request	Write.indication
Write.confirm	Write.response
UnconfirmedWrite.request	UnconfirmedWrite.indication
InformationReport.indication	InformationReport.request
DataNotification.indication	DataNotification.request
DataNotification.response	DataNotification.confirm
NOTE The DataNotification service can be used in application contexts using either SN referencing or LN referencing.	

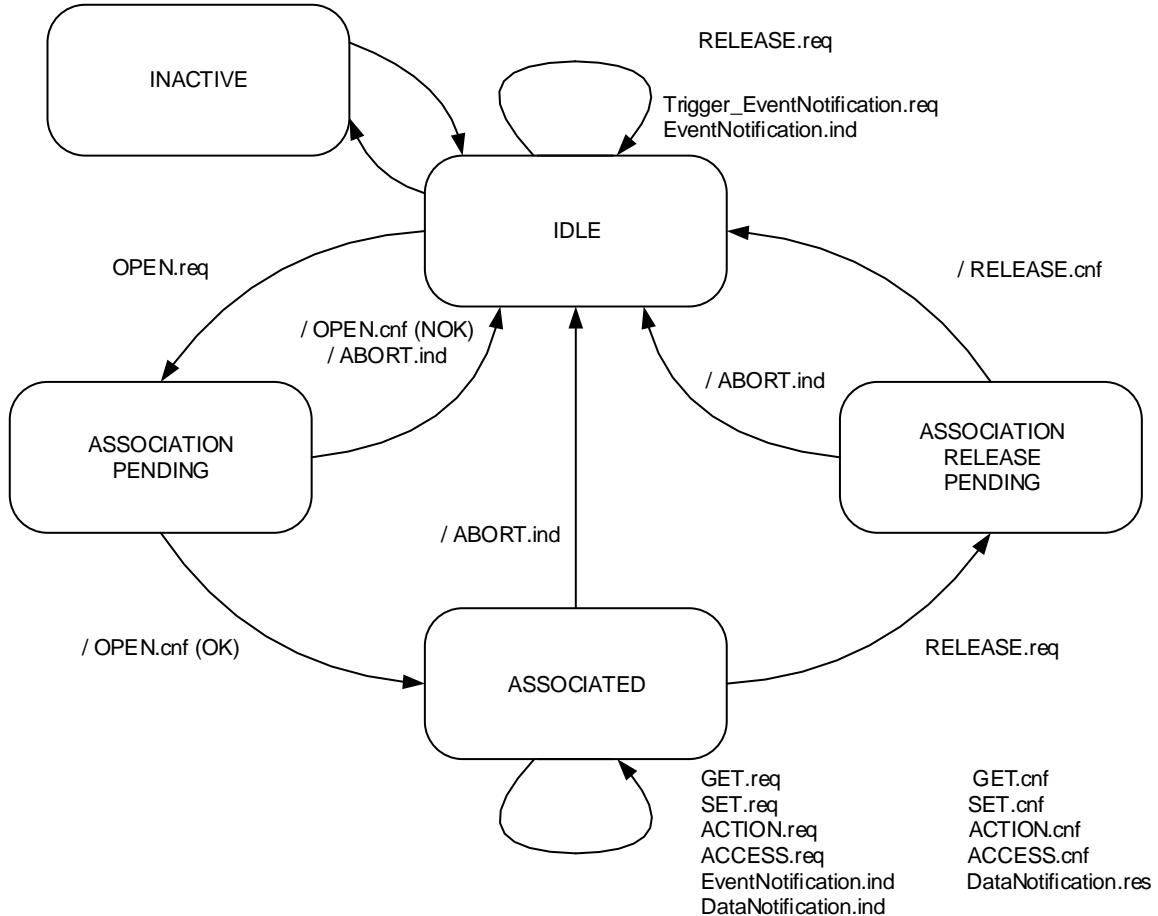
When the server uses SN referencing, a mapping between the service primitives using LN referencing and SN referencing takes place on the client side. This mapping is specified in 9.4.6.9, 9.4.6.10, 9.4.6.11 and 9.4.6.12.

## 9.4 DLMS/COSEM application layer protocol specification

### 9.4.1 The control function (CF)

#### 9.4.1.1 State definitions of the client side control function

Figure 114 shows the state machine for the client side CF, see also Figure 86.



NOTE 1 On the state diagrams of the client and server CF, the following conventions are used:

- service primitives with no “/” character as first character are “stimulants”: the invocation of these primitives is the origin of the state transition;
- service primitives with an “/” character as first character are “outputs”: the generation of these primitives is done on the state transition path.

**Figure 114 – Partial state machine for the client side control function**

The state definitions of the client CF – and of the AL including the CF – are as follows:

**INACTIVE** In this state, the CF has no activity at all: it neither provides services to the AP nor uses services of the supporting protocol layer.

**IDLE** This is the state of the CF when there is no AA existing, being released, or being established<sup>7</sup>. Nevertheless, some data exchange between the client and server – if the physical channel is already established – is possible. The CF can handle the EventNotification service.

NOTE 2 State transitions between the INACTIVE and IDLE states are controlled outside of the protocol. For example, it can be considered that the CF makes the state transition from INACTIVE to IDLE by being instantiated and bound on the top of the supporting protocol layer. The opposite transition may happen by deleting the given instance of the CF.

<sup>7</sup> Note, that it is the state machine for the AL: lower layer connections, including the physical connection, are not taken into account. On the other hand, physical connection establishment is done outside of the protocol.

ASSOCIATION PENDING	The CF leaves the IDLE state and enters this state when the AP requests the establishment of an AA by invoking the COSEM-OPEN.request primitive (OPEN.req). The CF may exit this state and enter either the ASSOCIATED state or return to the IDLE state, and generates the COSEM-OPEN.confirm primitive, (/OPEN.cnf(OK)) or (/OPEN.cnf(NOK)), depending on the result of the association request. The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).
ASSOCIATED	The CF enters this state when the AA has been successfully established. All xDLMS services and APDUs are available in this state. The CF remains in this state until the AP requests the release of the AA by invoking the COSEM-RELEASE.request primitive (RELEASE.req). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).
ASSOCIATION RELEASE PENDING	The CF leaves the ASSOCIATED state and enters this state when the AP requests the release of the AA by invoking the COSEM-RELEASE.request primitive (RELEASE.req). The CF remains in this state, waiting for the response to this request from the server. As the server is not allowed to refuse a release request, after exiting this state, the CF always enters the IDLE state. The CF may exit this state by generating the COSEM-RELEASE.confirm primitive following the reception of a response from the server or by generating it locally (/RELEASE.cnf). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).

#### 9.4.1.2 State definitions of the server side control function

Figure 115 shows the state machine for the server side CF, see Figure 86.

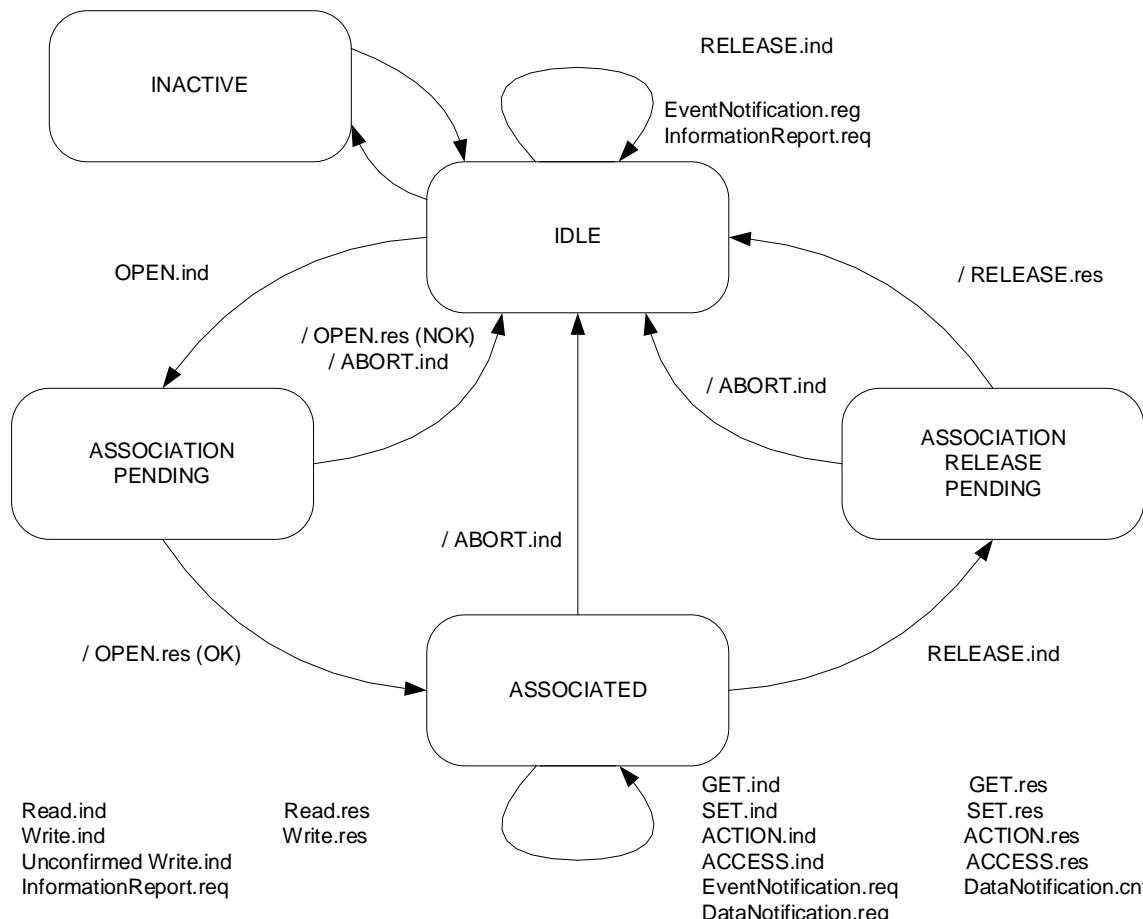


Figure 115 – Partial state machine for the server side control function

INACTIVE	In this state, the CF has no activity at all: it neither provides services to the AP nor uses services of the supporting protocol layer.
IDLE	This is the state of the CF when there is no AA existing, being released, or being established <sup>8</sup> . Nevertheless, some data exchange between the client and server – if the physical channel is already established – is possible. The CF can handle the EventNotification / InformationReport services.
ASSOCIATION PENDING	The CF leaves the IDLE state and enters this state when the client requests the establishment of an AA, and the server AL generates the COSEM-OPEN.indication primitive (/OPEN.ind). The CF may exit this state and enter either the ASSOCIATED state or return to the IDLE state, depending on the result of the association request, and invokes the COSEM-OPEN.response primitive, (/OPEN.res(OK)) or (/OPEN.res(NOK)). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).
ASSOCIATED	The CF enters this state when the AA has been successfully established. All xDLMS services and APDUs are available in this state. The CF remains in this state until the client requests the release of the AA, and the server AL generates the COSEM-RELEASE.ind primitive (/RELEASE.ind). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).
ASSOCIATION RELEASE PENDING	The CF leaves the ASSOCIATED state and enters this state when the client request the release of an AA, and the server AP receives the COSEM-RELEASE.indication primitive (/RELEASE.ind). The CF remains in this state, waiting that the AP accepts the release request. As the server is not allowed to refuse a release request, after exiting this state, the CF always enters the IDLE state. The CF may exit this state when the AP accepts the release of the AA, and invokes the COSEM-RELEASE.response primitive (RELEASE.res). The CF also exits this state and returns to the IDLE state with generating the COSEM-ABORT.indication primitive (/ABORT.ind).

#### 9.4.2 The ACSE services and APDUs

##### 9.4.2.1 ACSE functional units, services and service parameters

The DLMS/COSEM AL ACSE is based on the connection-oriented ACSE, as specified in ISO/IEC 15953:1999 and ISO/IEC 15954:1999.

Functional units are used to negotiate ACSE user requirements during association establishment. Five functional units are defined:

- Kernel functional unit;
- Authentication functional unit;
- ASO-context negotiation functional unit;

NOTE 1 ISO/IEC 15953:1999 and ISO/IEC 15954:1999 use the term ‘ASO-context’. In DLMS/COSEM the term ‘Application context’ is used as in ISO/IEC 8649 / ISO/IEC 8650.

- Higher Level Association functional unit; and
- Nested Association functional unit.

The DLMS/COSEM AL uses only the Kernel and the Authentication functional unit.

The acse-requirements parameters of the AARQ and AARE APDUs are used to select the functional units for the association.

<sup>8</sup> Note that it is the state machine for the AL: lower layer connections, including the physical connection, are not taken into account. On the other hand, physical connection establishment is done outside of the protocol.

The Kernel functional unit is always available and includes the basic services A-ASSOCIATE, A-RELEASE.

The Authentication functional unit supports authentication during association establishment. The availability of this functional unit is negotiated during association establishment. This functional unit does not include additional services. It adds parameters to the A-ASSOCIATE service.

Table 81 shows the services, APDUs and APDU fields associated with the ACSE functional units, as used by the DLMS/COSEM AL. The abstract syntax of the ACSE APDUs is specified in 9.5.

**Table 81 – Functional Unit APDUs and their fields**

Functional unit	Service	APDU	Field name	Presence
Kernel	A-ASSOCIATE	AARQ	protocol-version application-context-name called-AP-title called-AE-qualifier called-AP-invocation-identifier called-AE-invocation-identifier calling-AP-title calling-AE-qualifier calling-AP-invocation-identifier calling-AE-invocation-identifier implementation-information user-information <sup>2)</sup> (carrying an xDLMS Initiate.request APDU) dedicated-key response-allowed proposed-quality-of-service proposed-dlms-version-number proposed-conformance client-max-receive-pdu-size	O M U U U U U U U O M U U U M M M
		AARE	protocol-version application-context-name result result-source-diagnostic responding-AP-title responding-AE-qualifier responding-AP-invocation-identifier responding-AE-invocation-identifier implementation-information user-information <sup>3)</sup> (carrying an xDLMS initiateResponse APDU) negotiated-quality-of-service negotiated-dlms-version-number negotiated-conformance server-max-receive-pdu-size vaa-name (or carrying a confirmedServiceError APDU)	O M M M U U U U O M S U M M M M M S
	A-RELEASE	RLRQ	reason user-information	U U
		RLRE	reason user-information	U U

Functional unit	Service	APDU	Field name	Presence
Authentication	A-ASSOCIATE	AARQ	sender-acse-requirements mechanism-name calling-authentication-value	U U U
		AARE	responder-acse-requirements mechanism-name responding-authentication-value	U U U
NOTE 1 This table is based on ISO/IEC 15954:1999, Table 2 and 3. The fields are listed in the order as they are in the ACSE APDUs.				
M Presence is mandatory O Presence is ACM option U Presence is ACSE service-user option S The parameter is selected among other S-parameters as internal response of the server ASE environment.				
NOTE 2 According to ISO/IEC 15953:1999 the user-information parameter is optional. However, in the DLMS/COSEM environment it is mandatory in the AARQ / AARE APDUs.				
There are several changes in ISO/IEC 15953:1999 and ISO/IEC 15954:1999 compared to ISO/IEC 8649 and ISO/IEC 8650-1: <ul style="list-style-type: none"> <li>- In ISO/IEC 15954, protocol-version is mandatory in the AARQ and optional in the AARE. In DLMS/COSEM it is kept as mandatory for backward compatibility;</li> <li>- Instead of "application-context-name", "ASO-context-name" is used. In DLMS/COSEM, "application-context-name" is kept. ISO/IEC15954 7.1.5.2 specifies this: the ASO-context-name is optional. If backward compatibility with older implementations of ACSE is desired, it must be present. Therefore, in DLMS/COSEM it is mandatory;</li> <li>- In ISO/IEC 15954, the result and result-source-diagnostic parameters are optional. ISO/IEC 15954 7.1.5.8 and 7.1.5.9 specifies this: The Result / Result-source-diagnostic are optional. If backward compatibility with older implementations of ACSE is desired, it must be present. Therefore, in DLMS/COSEM these parameters are mandatory.</li> </ul>				

In general, the value of each field of the AARQ APDU is determined by the parameters of the COSEM-OPEN.request service primitive. Similarly, the value if each field of the AARE is determined by the COSEM-OPEN.response primitive. The COSEM-OPEN service is specified in 9.3.2.

The fields of the AARQ and AARE APDU are specified below. Managing these fields is specified in 9.4.4.1.

- protocol-version: the DLMS/COSEM AL uses the default value version 1. For details see ISO/IEC 15954:1999;
- application-context-name: COSEM application context names are specified in 9.4.2.2.2;  
NOTE 2 ISO/IEC 15953:1999 and ISO/IEC 15954:1999 use "ASO-context-name"
- called-, calling- and responding- titles, qualifiers and invocation-identifiers: these optional fields carry the value of the respective parameters of the COSEM-OPEN service. For details see ISO/IEC 15954:1999;
- implementation-information: this field is not used by the DLMS/COSEM AL. For details see ISO/IEC 15954:1999;
- user-information: in the AARQ APDU, it carries an xDLMS InitiateRequest APDU holding the elements of the Proposed\_xDLMS\_Context parameter of the COSEM-OPEN.request service primitive. In the AARE APDU, it carries an xDLMS InitiateResponse APDU, holding the elements of the Negotiated\_xDLMS\_Context parameter, or an xDLMS confirmedServiceError APDU, holding the elements of the xDLMS\_Initiate\_Error parameter of the COSEM-OPEN.response service primitive;
- sender- and responder-acse-requirements: this field is used to select the optional functional units of the AARQ / AARE. In COSEM, only the Authentication functional unit is used. When present, it carries the value of BIT STRING { authentication (0) }. Bit set: authentication functional unit selected;
- mechanism-name: COSEM authentication mechanism names are specified in 9.4.2.2.3;
- calling- and responding- authentication-value: see 9.2.2.2.2;

- result: the value of this field is determined by the COSEM AP (acceptor) or the DLMS/COSEM AL (ACPM) as specified below. It is used to determine the value of the Result parameter of the COSEM-OPEN.confirm primitive:
  - if the AARQ APDU is rejected by the ACPM (i.e. the COSEM-OPEN.indication primitive is not issued by the DLMS/COSEM AL), the value “rejected (permanent)” or “rejected (transient)” is assigned by the ACPM;
  - otherwise, the value is determined by the Result parameter of the COSEM-OPEN.response APDU;
- result-source-diagnostic: this field contains both the Result source value and the Diagnostic value. It is used to determine the value of the Failure\_Type parameter of the COSEM-OPEN.confirm primitive:
  - Result-source value: if the AARQ is rejected by the ACPM, (i.e. the COSEM-OPEN.indication primitive is not issued by the DLMS/COSEM AL) the ACPM assigns the value “ACSE service-provider”. Otherwise, the ACPM assigns the value “ACSE service-user”;
  - Diagnostic value: If the AARQ is rejected by the ACPM, the appropriate value is assigned by the ACPM. Otherwise, the value is determined by the Failure\_Type parameter of the COSEM-OPEN.response primitive. If the Diagnostic parameter is not included in the .response primitive, the ACPM assigns the value “null”.

The parameters of the RLRQ / RLRE APDUs – used when the COSEM-RELEASE service (see 9.3.3) is invoked with the parameter Use\_RLRQ\_RLRE == TRUE – are specified below.

- reason: carries the appropriate value as specified in 9.3.2;
- user-information: if present, it carries an xDLMS InitiateRequest / InitiateResponse APDU, holding the elements of the Proposed\_xDLMS\_Context / Negotiated\_xDLMS\_Context parameter of the COSEM-RELEASE.request / .response service primitive respectively. See 9.3.2.

#### 9.4.2.2 Registered COSEM names

##### 9.4.2.2.1 General

Within an OSI environment, many different types of network objects must be identified with globally unambiguous names. These network objects include abstract syntaxes, transfer syntaxes, application contexts, authentication mechanism names, etc. Names for these objects in most cases are assigned by the committee developing the particular basic ISO standard or by implementers' workshops, and should be registered. For DLMS/COSEM, these object names are assigned by the DLMS UA, and are specified below.

The decision no. 1999.01846 of OFCOM, Switzerland, attributes the following prefix for object identifiers specified by the DLMS User Association.

{ joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) }
--

NOTE As specified in ITU-T X.660 A.2.4, for historical reasons, the secondary identifiers ccitt and joint-iso-ccitt are synonyms for itu-t and joint-iso-itu-t, respectively, and thus may appear in ASN.1 OBJECT IDENTIFIER values, and also identify the corresponding primary integer value.
---

For DLMS/COSEM, object identifiers are specified for naming the following items:

- COSEM application context names;
- COSEM authentication mechanism names;
- cryptographic algorithm ID-s.

##### 9.4.2.2.2 The COSEM application context

In order to effectively exchange information within an AA, the pair of AE-invocations shall be mutually aware of, and follow a common set of rules that govern the exchange. This common set of rules is called the application context of the AA. The application context that applies to an AA is determined during its establishment.

An AA has only one application context. However, the set of rules that make up the application context of an AA may contain rules for alteration of that set of rules during the lifetime of the AA.

The following methods may be used:

- identifying a pre-existing application context definition;
- transferring an actual description of the application context.

In the COSEM environment, it is intended that an application context pre-exists and it is referenced by its name during the establishment of an AA. The application context name is specified as OBJECT IDENTIFIER ASN.1 type. COSEM identifies the application context name by the following object identifier value:

```
COSEM_Application_Context_Name ::=  
{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) application-context(1)  
context_id(x)}
```

The meaning of this general COSEM application context is:

- there are two ASEs present within the AE invocation, the ACSE and the xDLMS ASE;
- the xDLMS ASE is as it is specified in IEC 61334-4-41:1996;

NOTE With the COSEM extensions to DLMS, see 9.1.4.

- the transfer syntax is A-XDR.

The specific context\_id-s and the use of ciphered and unciphered APDUs are shown in Table 82:

**Table 82 – COSEM application context names**

Application context name	context_id	Unciphered APDUs	Ciphered APDUs
Logical_Name_Refencing_No_Ciphering ::=	context_id(1)	Yes	No
Short_Name_Refencing_No_Ciphering ::=	context_id(2)	Yes	No
Logical_Name_Refencing_With_Ciphering ::=	context_id(3)	Yes	Yes
Short_Name_Refencing_With_Ciphering ::=	context_id(4)	Yes	Yes

In order to successfully establish an AA, the application-context-name parameter of the AARQ and AARE APDUs should carry one of the “valid” names. The client proposes an application context name using the Application\_Context\_Name parameter of the COSEM-OPEN.request primitive. The server may return any value; either the value proposed or the value it supports.

#### 9.4.2.2.3 The COSEM authentication mechanism name

Authentication of the client, the server or both is one of the security aspects addressed by the DLMS/COSEM specification. Three authentication security levels are specified:

- no security (Lowest Level Security) authentication, see 9.2.2.2.2;
- Low Level Security (LLS) authentication, see 9.2.2.2.2.3;
- High Level Security (HLS) authentication, see 9.2.2.2.2.4.

DLMS/COSEM identifies the authentication mechanisms by the following general object identifier value:

```
COSEM_Authentication_Mechanism_Name ::=  
{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8)  
authentication_mechanism_name(2) mechanism_id(x)}
```

The value of the mechanism\_id element selects one of the security mechanisms specified:

**Table 83 – COSEM authentication mechanism names**

COSEM_lowest_level_security_mechanism_name ::=	mechanism_id(0)
COSEM_low_level_security_mechanism_name ::=	mechanism_id(1)
COSEM_high_level_security_mechanism_name ::=	mechanism_id(2)
COSEM_high_level_security_mechanism_name_using_MD5 ::=	mechanism_id(3)
COSEM_high_level_security_mechanism_name_using_SHA-1 ::=	mechanism_id(4)
COSEM_high_level_security_mechanism_name_using_GMAC ::=	mechanism_id(5)
COSEM_high_level_security_mechanism_name_using_SHA-256 ::=	mechanism_id(6)
COSEM_high_level_security_mechanism_name_using_ECDSA ::=	mechanism_id(7)

NOTE 1 With mechanism\_id(2), the method of processing the challenge is secret.  
 NOTE 2 The use of authentication mechanisms 3 and 4 are not recommended for new implementations.

When the Authentication\_Mechanism\_Name is present in the COSEM-OPEN service, the authentication functional unit of the A-ASSOCIATE service shall be selected. The process of LLS and HLS authentication is described in 9.2.2.2.2 and in 9.2.7.3.

#### 9.4.2.2.4 Cryptographic algorithm ID-s

Cryptographic algorithm IDs identify the algorithm for which a derived secret symmetrical key will be used. See 9.2.3.4.6.5.

Cryptographic algorithms are identified by the following general object identifier value:

COSEM_Cryptographic_Algorithm_Id ::=
{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) cryptographic-algorithms (3) algorithm_id(x)}

The values of the algorithm\_id-s are shown in Table 84. See also Table 26.

**Table 84 – Cryptographic algorithm ID-s**

COSEM_cryptographic_algorithm_name_aes-gcm-128 ::=	algorithm_id(0)
COSEM_cryptographic_algorithm_name_aes-gcm-256 ::=	algorithm_id(1)
COSEM_cryptographic_algorithm_name_aes-wrap-128 ::=	algorithm_id(2)
COSEM_cryptographic_algorithm_name_aes-wrap-256 ::=	algorithm_id(3)

#### 9.4.3 APDU encoding rules

##### 9.4.3.1 Encoding of the ACSE APDUs

The ACSE APDUs shall be encoded in BER (ISO/IEC 8825-1:2015). The user-information parameter of these APDUs shall carry the xDLMS InitiateRequest / InitiateResponse / confirmedServiceError APDU as appropriate, encoded in A-XDR, and then encoding the resulting OCTET STRING in BER.

Examples for AARQ/AARE APDU encoding are given in Clauses 11 and 12.

##### 9.4.3.2 Encoding of the xDLMS APDUs

The xDLMS APDUs shall be encoded in A-XDR, as specified in IEC 61334-6:2000.

##### 9.4.3.3 XML

Depending on the parametrization of the “Push setup” object the DataNotification APDU can be encoded as an XML document using the XML schema specified in 9.6.

NOTE The use of XML to encode the other APDUs is not in the Scope of this Technical Report.

312/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

#### 9.4.4 Protocol for application association establishment

##### 9.4.4.1 Protocol for the establishment of confirmed application associations

AA establishment using the A-Associate service of the ACSE is the key element of DLMS/COSEM interoperability. The participants of an AA are:

- a client AP, proposing an AA; and
- a server AP, accepting the proposed AA or not.

NOTE 1 To support multicast and broadcast services, an AA can also be established between a client AP and a group of server APs.

Figure 116 gives the MSC for the case, when:

- the COSEM-OPEN.request primitive requests a confirmed AA;
- the connection of the supporting protocol layers is required for the establishment of this AA.

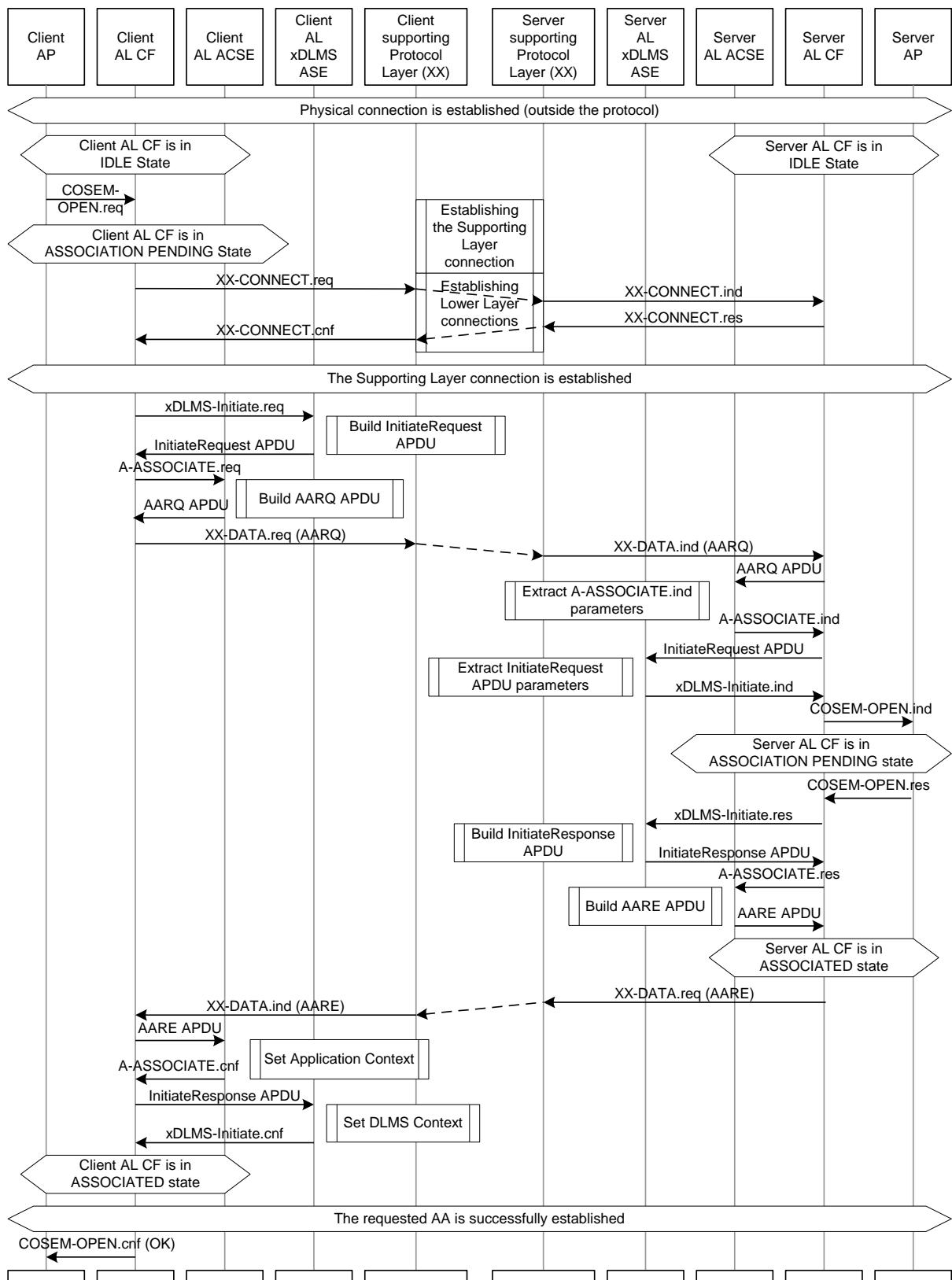
A client AP that desires to establish a confirmed AA, invokes the COSEM-OPEN.request primitive of the ASO with Service\_Class == Confirmed. Note, that the PH layer has to be connected before the COSEM-OPEN service is invoked. The response-allowed parameter of the xDLMS InitiateRequest APDU is set to TRUE. The client AL waits for an AARE APDU, prior to generating the .confirm primitive, with a positive – or negative – result.

The client CF enters the ASSOCIATION PENDING state. It examines then the Protocol\_Connection\_Parameters parameter. If this indicates that the establishment of the supporting protocol layer connection is required, it establishes the connection. The CF assembles then –with the help of the xDLMS ASE and the ACSE – the AARQ APDU containing the parameters of the COSEM-OPEN.request primitive received from the AP and sends it to the server.

The CF of the server AL gives the AARQ APDU received to the ACSE. It extracts the ACSE related parameters then gives back the control to the CF. The CF passes then the contents of the user-information parameter of the AARQ APDU – carrying an xDLMS InitiateRequest APDU – to the xDLMS ASE. It retrieves the parameters of this APDU, then gives back the control to the CF. The CF generates the COSEM-OPEN.indication to the server AP with the parameters received APDU and enters the ‘ASSOCIATION PENDING’ state.

NOTE 2 Some service parameters of the COSEM-OPEN.indication primitive (address information, User\_Information) do not come from the AARQ APDU, but from the supporting protocol layer frame carrying the AARQ APDU. In some communication profiles, the Service\_Class parameter of the COSEM-OPEN service is linked to the frame type of the supporting protocol layer. In some other communication profiles, it is linked to the response-allowed field of the xDLMS Initiate.request APDU. See also 10.

NOTE 3 The ASEs only extract the parameters; their interpretation and the decision whether the proposed AA can be accepted or not is the job of the server AP.



**Figure 116 – MSC for successful AA establishment preceded by a successful lower layer connection establishment**

The server AP parses the fields of the AARQ APDU as described below.

Fields of the Kernel functional unit:

- application-context-name: it carries the COSEM\_Application\_Context\_Name the client proposes for the association;
- calling-AP-title: when the proposed application context uses ciphering, it shall carry the client system title. If a client system title has already been sent during a registration process, like in the S-FSK PLC profile, the calling-AP-title field should carry the same system title. Otherwise, the AA should be rejected and appropriate diagnostic information should be sent;
- calling\_AE\_invocation\_identifier: this field supports the client user identification process; see DLMS UA 1000-1;
- calling-AE-qualifier: This field can be used to transport the public key certificate of the digital signature key of the client.

Fields of the authentication functional unit (when present):

- sender-acse-requirements:
  - a) if not present or present but bit 0 = 0, then the authentication functional unit is not selected. Any following fields of the authentication functional unit may be ignored;
  - b) if present and bit 0 = 1 then the authentication functional unit is selected;
- mechanism-name: it carries the COSEM\_Authentication\_Mechanism\_Name the client proposes for the association;
- calling-authentication-value: it carries the authentication value generated by the client.

If the value of the mechanism-name or the calling-authentication-value fields are not acceptable then the proposed AA shall be refused.

When the parsing of the fields of the Kernel and the authentication functional unit is completed, the server continues with parsing the parameters of the xDLMS InitiateRequest APDU, carried by the user-information field of the AARQ:

- dedicated-key: it carries the dedicated key to be used in the AA being established;
- response-allowed: If the proposed AA is confirmed and the value of this parameter is TRUE (default), the server shall send back an AARE APDU. Otherwise, the server shall not respond. See also Clause 10;
- proposed-dlms-version-number, see 9.1.4.6;
- proposed-conformance, see 9.4.6.1;
- client-max-receive-pdu-size, see 9.1.4.8.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	315/614
-----------------------	------------	-----------------------	---------

If all elements of the proposed AA are acceptable, the server AP invokes the COSEM-OPEN.response service primitive with the following parameters:

- Application\_Context\_Name: the same as the one proposed;
- Result: accepted;
- Failure\_Type: Result-source: acse-service-user; Diagnostic: null;
- Responding\_AP\_Title: if the negotiated application context uses ciphering, it shall carry the server system title. If a server system title has already been sent during a registration process, like in the case of the S-FSK PLC profile, the Responding\_AP\_Title parameter should carry the same system title. Otherwise, the AA should be aborted by the client;
- Responding\_AE\_Qualifier: This field can be used to transport the public key certificate of the digital signature key of the server;
- Fields of the AARE authentication functional unit:
  - (Responder\_)ACSE\_Requirements:
    - i) when no security (Lowest Level Security) authentication or Low Level Security (LLS) authentication is used, this field shall not be present, or if present, bit 0 (authentication) shall be set to 0. Any following fields of the authentication functional unit may be ignored;
    - ii) when High Level Security (HLS) authentication is used, this field shall be present and bit 0 (authentication) shall be set to 1;
  - Security\_Mechanism\_Name: it shall carry the COSEM\_Authentication\_Mechanism\_Name negotiated;
  - Responding\_Authentication\_Value: it carries the authentication value generated by the server (StoC).
- Negotiated\_xDLMS\_Context.

The CF assembles the AARE APDU – with the help of the xDLMS ASE and the ACSE – and sends it to the client AL via the supporting layer protocols, and enters the ASSOCIATED state. The proposed AA is established now; the server is able to receive xDLMS data transfer service request(s) – both confirmed and unconfirmed – and to send responses to confirmed service requests within this AA.

At the client side, the fields of the AARE APDU received are extracted with the help of the ACSE and the xDLMS ASE, and passed to the client AP via the COSEM-OPEN.confirm service primitive. At the same time, the client AL enters the ‘ASSOCIATED’ state. The AA is established now with the application context and xDLMS context negotiated.

If the application context proposed by the client is not acceptable or the authentication of the client is not successful, the COSEM-OPEN.response primitive is invoked with the following parameters:

- Application\_Context\_Name: the same as the one proposed, or the one supported by the server;
- Result: rejected-permanent or rejected-transient;
- Failure\_Type: Result-source: acse-service-user; Diagnostic: an appropriate value;
- User\_Information: an xDLMS InitiateResponse APDU with the parameters of the xDLMS context supported by the server.

If the application context proposed by the client is acceptable and the authentication of the client is successful but the xDLMS context cannot be accepted, the COSEM-OPEN.response primitive shall be invoked with the following parameters:

- Application\_Context\_Name: the same as the one proposed;
- Result: rejected-permanent or rejected-transient;
- Failure\_Type: Result-source: acse-service-user; Diagnostic: no-reason-given;
- xDLMS\_Initiate\_Error, indicating the reason for not accepting the proposed xDLMS context.

In these two cases, upon the invocation of the .response primitive, the CF assembles and sends the AARE APDU to the client via the supporting layer protocols. The proposed AA is not established, the server CF returns to the IDLE state.

At the client side, the fields of the AARE APDU received are extracted with the help of the ACSE and the xDLMS ASE, and passed to the client AP via the COSEM-OPEN.confirm primitive. The proposed AA is not established, the client CF returns to the IDLE state.

The server ACSE may not be capable of supporting the requested association, for example if the AARQ syntax or the ACSE protocol-version are not acceptable. In this case, it returns a COSEM-OPEN.response primitive to the client with an appropriate Result parameter. The result-source-diagnostic field of the AARE APDU is appropriately assigned the symbolic value of "acse-service-provider". The COSEM-OPEN.indication primitive is not issued. The association is not established.

#### **9.4.4.2 Repeated COSEM-OPEN service invocations**

If a COSEM-OPEN.request primitive is invoked by the client AP referring to an already established AA, then the AL locally and negatively confirms this request with the reason that the requested AA already exists. Note, that this is always the case for pre-established AAs. See 9.4.4.4.

If, nevertheless, a server AL receives an AARQ APDU referencing to an already existing AA, it simply discards this AARQ, or, if it is implemented, it may also respond with the optional exception-response APDU.

#### **9.4.4.3 Establishment of unconfirmed application associations**

A client AP that desires to establish an unconfirmed AA, invokes the COSEM-OPEN.request primitive of the ASO with Service\_Class == Unconfirmed. The response-allowed parameter of the xDLMS InitiateRequest APDU, carried by the user-information parameter of the AARQ is set to FALSE. The client AL does not wait any response from the server: the .confirm primitive is locally generated. Otherwise the procedure is the same as in the case of the establishment of confirmed AAs.

As the establishment of unconfirmed AAs does not require the server AP to respond to the association request coming from the client, in some cases – for example in the case of one-way communications or broadcasting – the establishment of unconfirmed AA is the only possibility.

After the establishment of an unconfirmed AA, xDLMS data transfer services using LN referencing can be invoked only in an unconfirmed manner, until the association is released. With SN referencing, only the UnconfirmedWrite service can be used.

#### **9.4.4.4 Pre-established application associations**

The purpose of pre-established AAs is to simplify data exchange. The AA establishment and release phases (phases 1 and 3 on Figure 4), using the COSEM-OPEN and COSEM-RELEASE services are eliminated and only data transfer services are used. This Technical Report does not specify how to establish such AAs: it can be considered, that this has already been done. Pre-established AAs can be considered to exist from the moment the lower layers are able to transmit APDUs between the client and the server.

As for all AAs, the logical devices shall contain an Association LN / SN interface object for the pre-established associations, too.

A pre-established AA can be either confirmed or unconfirmed (depending on the way it has been pre-established).

A pre-established AA cannot be released.

### **9.4.5 Protocol for application association release**

#### **9.4.5.1 Overview**

An existing AA can be released gracefully or non-gracefully. Graceful release is initiated by the client AP. Non-graceful release takes place when an event unexpected by the AP occurs, for example a physical disconnection is detected.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	317/614
-----------------------	------------	-----------------------	---------

#### 9.4.5.2 Graceful release of an application association

DLMS/COSEM provides two mechanisms to release AAs:

- by disconnecting the supporting protocol layer of the AL;
- by using the ACSE A-Release service.

The first mechanism shall be supported in all profiles where the supporting protocol layer of the AL is connection oriented.

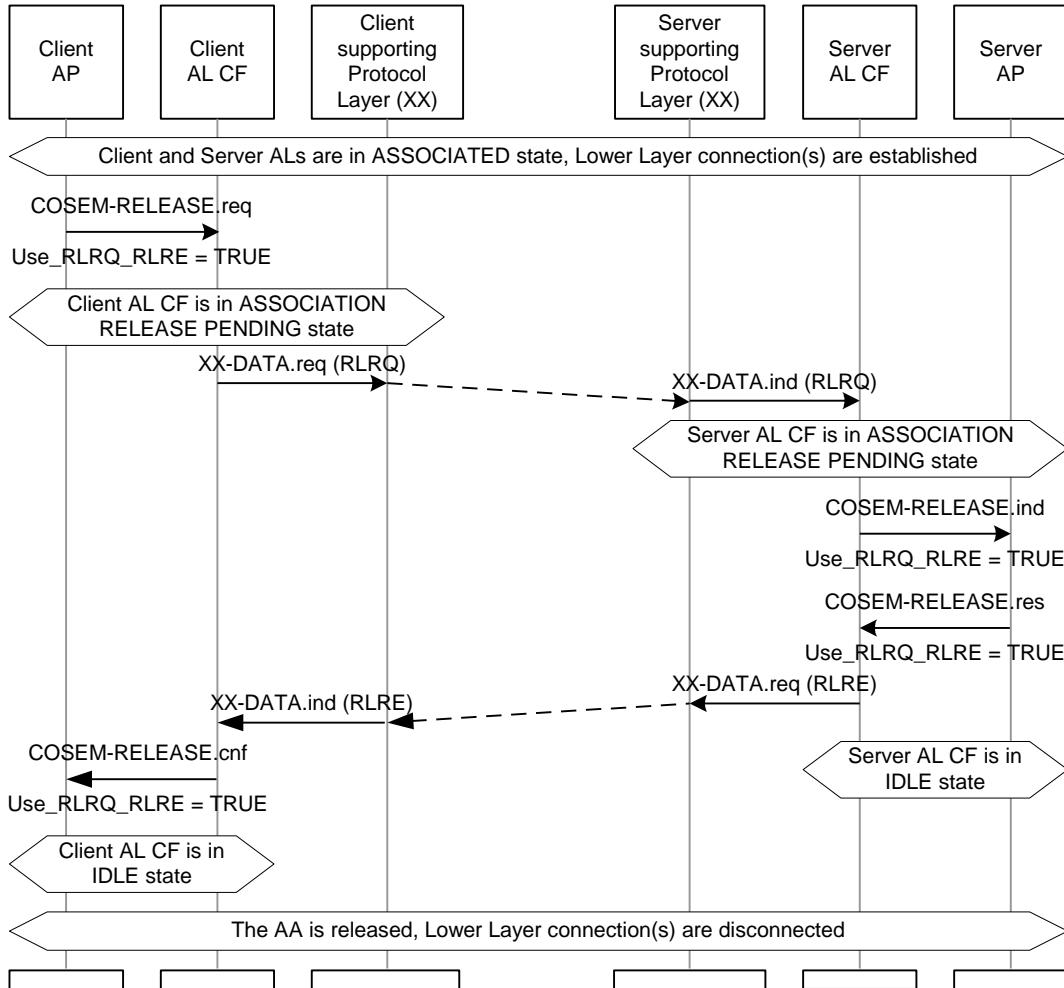
EXAMPLE The 3-layer, HDLC based, connection oriented profile.

To release an AA this way, the COSEM-RELEASE service shall be invoked with the Use\_RLRQ\_RLRE parameter not present or FALSE. Disconnecting the supporting protocol layer shall release all AAs built on that supporting protocol layer connection.

The second mechanism can be used to release an AA without disconnecting the supporting protocol layer. It shall be supported in all profiles when the supporting protocol layer is connectionless. It may be also used when the supporting protocol layer is connection-oriented but the connection is not managed by the AL, or disconnection of the supporting protocol layer is not practical because other applications may use it, or when there is a need to secure the COSEM-RELEASE service. It is the only way to release unconfirmed AAs.

To release an AA in this way, the COSEM-RELEASE service shall be invoked with the Use\_RLRQ\_RLRE parameter = TRUE. As specified in 9.3.3, the COSEM-RELEASE service can be secured by including a ciphered xDLMS InitiateRequest / InitiateResponse in the user-information field of the RLRQ / RLRE APDUs respectively, thus preventing a potential denial of service attack.

An example for releasing an AA using the ACSE A-RELEASE service is shown in Figure 117.



NOTE The release of an AA may require internal communication between the ASEs of the CF. These are not shown in Figure 117 to Figure 119.

**Figure 117 – Graceful AA release using the A-RELEASE service**

A client AP that desires to release an AA using the A-RELEASE service, shall invoke the COSEM-RELEASE.request service primitive with `Use_RLRQ_RLRE == TRUE`. The client CF enters the ASSOCIATION RELEASE PENDING state. It constructs then an RLRQ APDU and sends it to the server. If the AA to be released has been established with a ciphered context, then the user information field of the RLRQ APDU may contain a ciphered xDLMS InitiateRequest APDU. See 9.3.3.

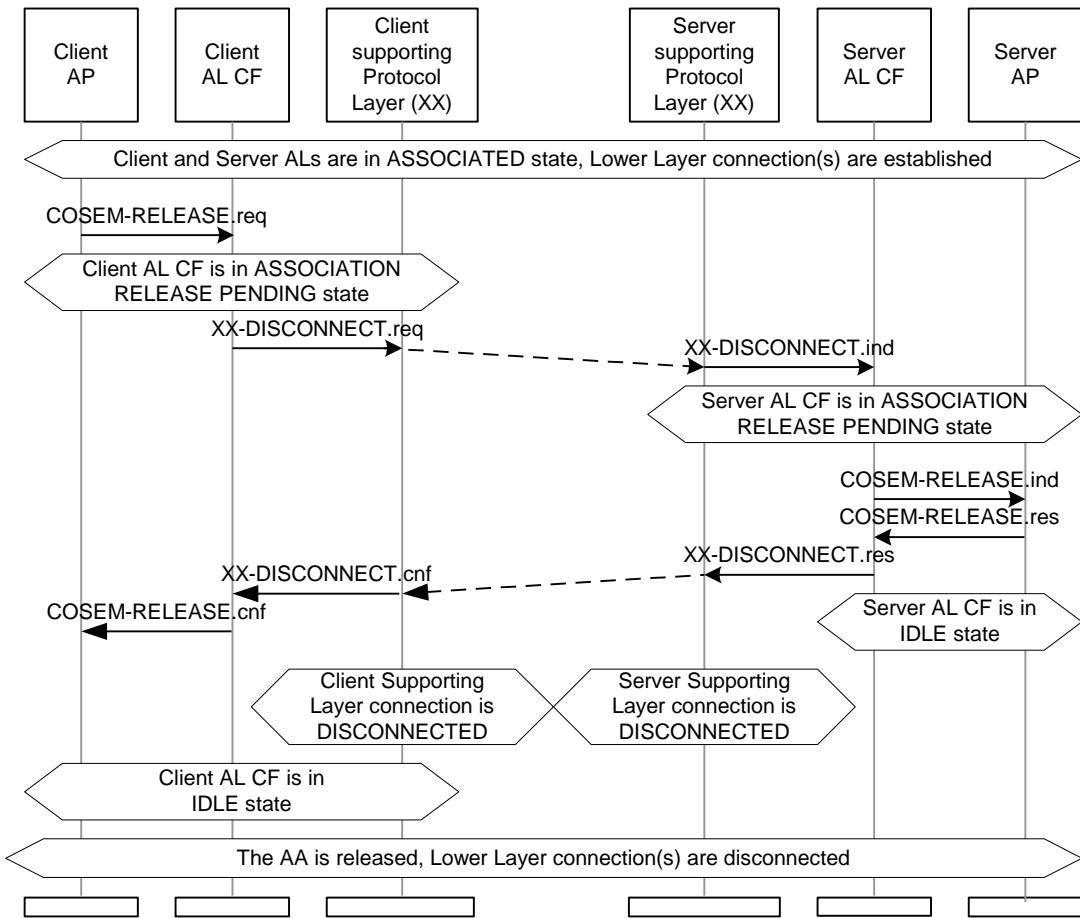
When the server AL CF receives the RLRQ APDU, it checks first if the user-information field contains a ciphered xDLMS InitiateRequest APDU. If so, it tries to decipher it. If this is successful, it enters the ASSOCIATION RELEASE PENDING state and generates a COSEM-RELEASE.indication primitive with `Use_RLRQ_RLRE == TRUE`. Otherwise, it silently discards the RLRQ APDU and stays in the ASSOCIATED state.

The .response primitive is invoked by the server AP to indicate if the release of the AA is accepted or not, but only if the AA to be released is confirmed. Note, that the server AP cannot refuse a release request. Upon the reception of the .response primitive the server AL CF constructs a RLRE APDU and sends it to the client. If the RLRQ APDU contained a ciphered xDLMS initiateRequest APDU then the RLRE ADU shall contain a ciphered xDLMS InitiateResponse APDU. The server AL CF returns to the IDLE state.

The .confirm primitive is generated by the client AL CF when the RLRE APDU is received. The supporting protocol layer is not disconnected. The client AL CF returns to the IDLE state.

If the RLRE APDU received contains a ciphered xDLMS InitiateResponse APDU but it cannot be deciphered, then the RLRE APDU shall be discarded. It is left to the client to cope with the situation.

Figure 118 gives an example of graceful releasing a confirmed AA by disconnecting the corresponding lower layer connection.



**Figure 118 – Graceful AA release by disconnecting the supporting protocol layer**

A client AP that desires to release an AA not using the A-RELEASE service, invokes the COSEM-RELEASE.request primitive with Use\_RLRQ\_RLRE == FALSE or not present. The client AL CF enters the ASSOCIATION RELEASE PENDING state.

In communication profiles where the RLRQ service is mandatory, invoking the .request primitive with no Use\_RLRQ\_RLRE or with Use\_RLRQ\_RLRE == FALSE may lead to an error: the .request shall be locally and negatively confirmed. The client AL CF returns to the IDLE state.

When the client AL CF receives the .request primitive, it sends an XX-DISCONNECT.request primitive to the server.

When the server AL CF receives the XX-DISCONNECT.request primitive, the CF enters the ASSOCIATION RELEASE PENDING state. The COSEM-RELEASE.indication primitive is generated by the server AL CF with Use\_RLRQ\_RLRE == FALSE or not present.

The COSEM-RELEASE.response primitive is invoked by the server AP to indicate if the release of the AA is accepted or not. Note, that the server AP cannot refuse a release request. Upon the reception of this primitive, the server AL CF sends an XX-DISCONNECT.response primitive to the client and returns to the IDLE state.

The COSEM-RELEASE.confirm primitive is generated by the client AL when the XX-DISCONNECT.confirm primitive is received. The supporting protocol layer is disconnected. The client AL CF returns to the IDLE state.

#### 9.4.5.3 Non-graceful release of an application association

Various events may result in a non-graceful release of an AA: detection of the disconnection of any lower layer connection (including the physical connection), detecting a local error, etc.

Non-graceful release – abort – of an AA is indicated to the COSEM AP with the help of the COSEM-ABORT service. The Diagnostics parameter of this service indicates the reason for the non-graceful AA release. The non-graceful release of AA is not selective: if it happens, all the existing association(s) – except the pre-established ones – shall be aborted.

Figure 119 shows the message sequence chart for aborting the AA, due to the detection of a physical disconnection.

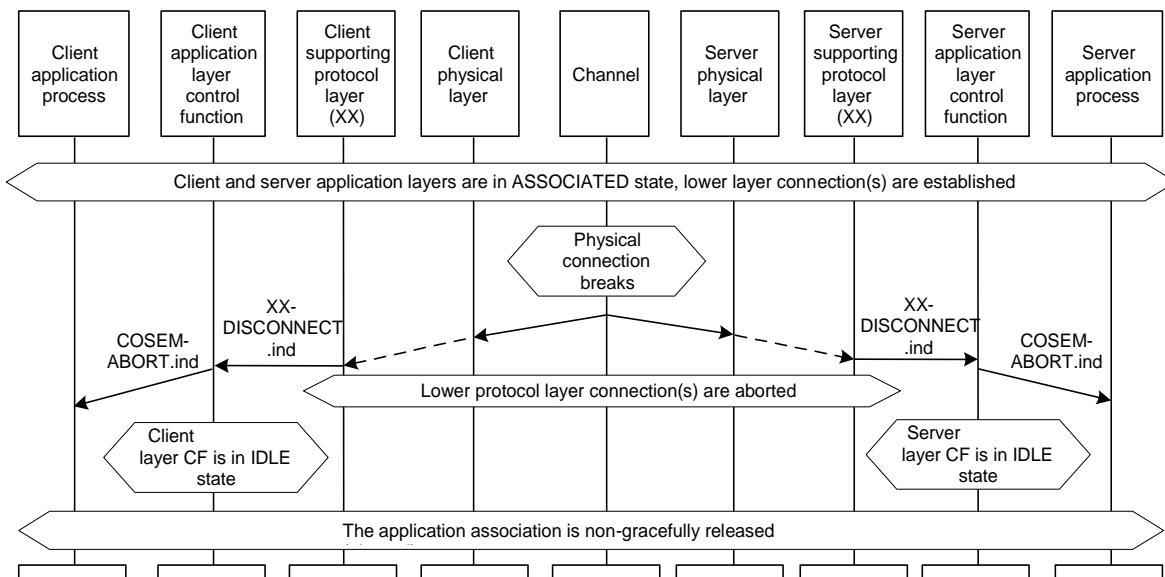


Figure 119 – Aborting an AA following a PH-ABORT.indication

#### 9.4.6 Protocol for the data transfer services

##### 9.4.6.1 Negotiation of services and options – the conformance block

The conformance block allows clients and servers using the same DLMS/COSEM protocol, but supporting different capabilities to negotiate a compatible set of capabilities so that they can communicate. It is carried by the DLMS\_Conformance parameter of the COSEM-OPEN service.

In DLMS/COSEM none of the services or options are mandatory: the ones to be used are negotiated via the COSEM-OPEN service (the proposed-conformance parameter of the xDLMS InitiateRequest APDU and the negotiated-conformance parameter of the xDLMS InitiateResponse APDU). An implemented service shall be fully conforming to its specification. If a service or option is not present in the negotiated conformance block, it should not be requested by the client.

The xDLMS conformance block can be distinguished from the DLMS conformance block specified in IEC 61334-4-41:1996 by its tag: APPLICATION 31. It is shown in Table 85.

**Table 85 – xDLMS Conformance block**

<b>Conformance block bit</b>	<b>Reserved</b>	<b>LN referencing</b>	<b>SN referencing</b>
0	x		
1		general-protection <sup>1</sup>	general-protection <sup>1</sup>
2		general-block-transfer	general-block-transfer
3			read
4			write
5			unconfirmed-write
6		delta-value-encoding	delta-value-encoding
7	x		
8		attribute0-supported-with-set	
9		priority-mgmt-supported	
10		attribute0-supported-with-get	
11		block-transfer-with-get-or-read	block-transfer-with-get-or-read
12		block-transfer-with-set-or-write	block-transfer-with-set-or-write
13		block-transfer-with-action	
14		multiple-references	multiple-references
15			information-report
16		data-notification	data-notification
17		access	
18			parameterized-access
19		get	
20		set	
21		selective-access	
22		event-notification	
23		action	

<sup>1</sup> general-protection includes general-glo-ciphering, general-ded-ciphering, general-ciphering and general-signing

#### 9.4.6.2 Confirmed and unconfirmed xDLMS service invocations

##### 9.4.6.2.1 Service invocations by the client

In general, xDLMS services may be invoked in a confirmed or an unconfirmed manner. The time sequence of the service primitives corresponds to:

- Figure 112 item a) in the case of confirmed service invocations; and
- Figure 112 item d) in the case of unconfirmed service invocations.

A client AP that requires access to an attribute or a method of a COSEM object invokes the appropriate .request service primitive. The client AL constructs the APDU corresponding to the .request primitive and sends it to the server.

Upon the receipt of the .indication primitive, the server AP checks whether the service can be provided or not (validity, client access rights, availability, etc.). If it can, it locally applies the service required on the corresponding “real” object. In the case of confirmed services, the server AP invokes the appropriate .response primitive. The server AL constructs the APDU corresponding to the .response primitive and sends it to the client. The client AL generates the .confirm primitive.

If a confirmed service request cannot be processed by the server AL – for example the request has been received without establishing an AA first, or the request is otherwise erroneous – it is either discarded, or when possible, the server AL responds with a confirmedServiceError APDU, or, when implemented, with an ExceptionResponse APDU. These APDUs may contain diagnostic information about the reason of not being able to process the request. They are defined in 9.5.

Within confirmed AAs xDLMS services can be invoked in a confirmed or unconfirmed manner.

Within unconfirmed AAs xDLMS services may be invoked in an unconfirmed manner only. With this, collisions due to potential multiple responses in the case of multicasting and/or broadcasting can be avoided.

In the case of unconfirmed services, three different kinds of destination addresses are possible: individual, group or broadcast. Depending on the destination address type, the receiving station shall handle incoming APDUS differently, as follows:

- XX-APDUs with an individual address of a COSEM logical device. If they are received within an established AA they shall be sent to the COSEM logical device addressed, otherwise they shall be discarded;
- XX-APDUs with a group address of a group of COSEM logical devices. These shall be sent to the group of COSEM logical devices addressed. However, the message received shall be discarded if there is no AA established between a client and the group of COSEM logical devices addressed;
- XX-APDUs with the broadcast address shall be sent to all COSEM logical devices addressed. However, the message received shall be discarded if there is no AA established between a client and the All-station address.

**NOTE** Unconfirmed AA-s between a client and a group of logical devices are established with a COSEM-OPEN service with Service\_Class == Unconfirmed and a group of logical device addresses (for example broadcast address).

#### 9.4.6.2.2 Service invocations by the server (unsolicited services)

The unsolicited services that may be invoked by the server are:

- InformationReport;
- EventNotification;
- DataNotification.

The InformationReport and the EventNotification services may be invoked only in an unconfirmed manner. The corresponding time sequence diagram is Figure 112 item d).

The DataNotification service may be invoked in three different ways:

- 1) unconfirmed, retry on supporting protocol layer failure;
- 2) unconfirmed, retry on missing supporting protocol layer confirmation;
- 3) confirmed, retry on missing confirmation.

The time sequence of the service primitives corresponds to:

- in the case 1) Figure 112 item d);
- in the case 2) Figure 112 item b);
- in the case 3) Figure 112 item a).

The protocol of the DataNotification service is described in 9.4.6.7.

#### 9.4.6.3 Protocol for the GET service

When the client AP desires to read the value of one or more COSEM object attributes, it uses the GET service.

As explained in 9.3.6, the encoded form of the request shall always fit in a single APDU.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	323/614
-----------------------	------------	-----------------------	---------

On the other hand, the result may be too long to fit in a single APDU. In this case, either the service-specific or the general block transfer mechanism may be used. It is negotiated via bit 2 or bit 11 of the conformance block, see 9.4.6.1.

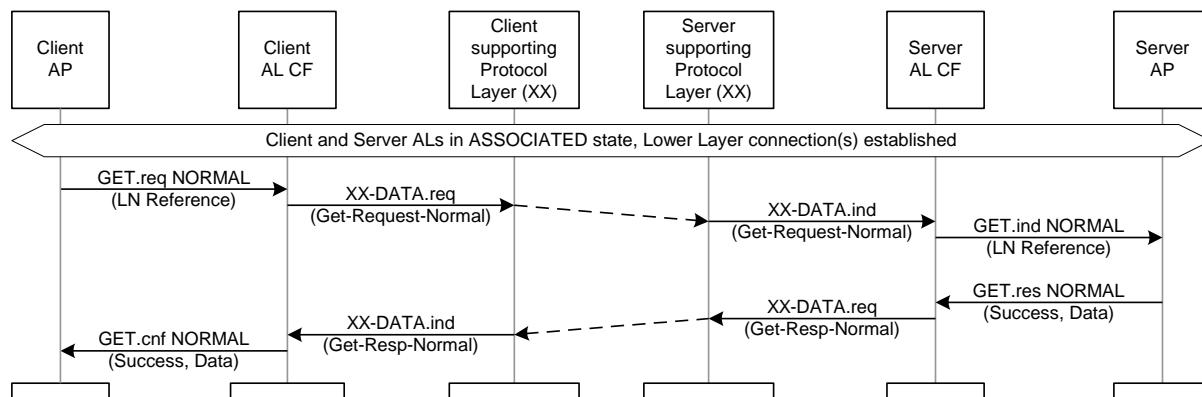
**NOTE** In some DLMS/COSEM communication profiles segmentation is available to transfer long APDUs.

The GET service primitive types and the corresponding APDUs are shown in Table 86.

**Table 86 – GET service types and APDUs**

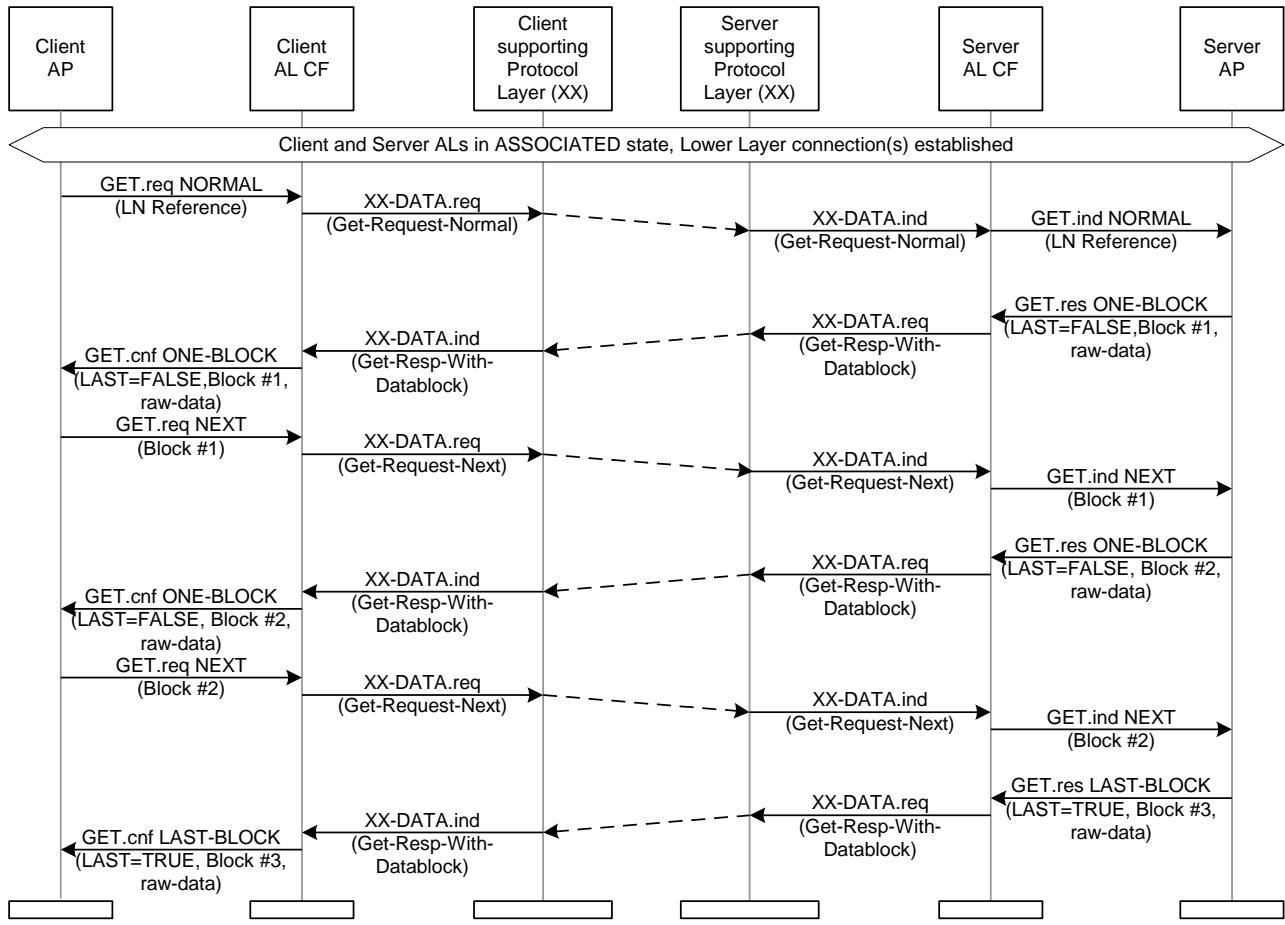
GET .req / .ind	Request APDU	Response APDU	GET .res / .cnf
NORMAL	Get-Request-Normal	Get-Response-Normal	NORMAL
		Get-Response-With-Datablock with Last-Block = FALSE	ONE-BLOCK
NEXT	Get-Request-Next	Get-Response-With-Datablock with Last-Block = FALSE	ONE-BLOCK
		Get-Response-With-Datablock with Last-Block = TRUE	LAST-BLOCK
WITH-LIST	Get-Request-With-List	Get-Response-With-List	WITH-LIST
		Get-Response-With-Datablock with Last-Block = FALSE	ONE-BLOCK

Figure 120 shows the MSC for a confirmed GET service in the case of success, without block transfer.



**Figure 120 – MSC of the GET service**

Figure 121 shows the MSC of a confirmed GET service in the case of success, with the result returned in three blocks using the service-specific block transfer mechanism.

**Figure 121 – MSC of the GET service with block transfer**

The GET.request primitive is invoked with Request\_Type == NORMAL or WITH-LIST as appropriate. As in this case the data to be returned is too long to fit in a single APDU, the server AP sends it in blocks. First, the data is encoded, as if it would fit into a single APDU:

- if the value of a single attribute was requested, only the type and value shall be encoded (Data). If the Data cannot be delivered, the response shall be of type GET-NORMAL with Data\_Access\_Result;
- if the value of a list of attributes was requested, then the list of results shall be encoded: for each attribute, either Data or Data\_Access\_Result.

The result is a series of bytes,  $B_1, B_2, B_3, \dots, B_N$ . The server may generate the complete response upon the receipt of the first GET.indication primitive or dynamically (on the fly).

The server AP assembles then a DataBlock\_G structure:

- Last\_Block == FALSE;
- Block\_Number == 1;
- Result (Raw\_Data) == the first K bytes of the encoded data:  $B_1, B_2, B_3, \dots, B_K$ .

It is recommended to start the numbering of the blocks from 1.

The server AP invokes then the GET-RESPONSE-ONE-BLOCK service primitive with Response\_Type == ONE-BLOCK carrying this DataBlock-G structure.

Upon the reception of the .response primitive, the server AL builds a Get-Response-With-Datablock APDU carrying the parameters of the .response primitive and sends it to the client.

Upon the reception of this APDU, the client AL generates a .confirm primitive with Response\_Type == ONE-BLOCK. The client AP is informed now that the response is provided in several blocks. It stores

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	325/614
-----------------------	------------	-----------------------	---------

the data block received ( $B_1, B_2, B_3, \dots, B_k$ ), then acknowledges its reception and asks for the next one by invoking a GET-REQUEST-NEXT service primitive. The block number shall be the same as the block number of the data block received. The client AL builds a Get-Request-Next APDU and sends it to the server.

When the server AL invokes the GET.indication primitive with Request\_Type == NEXT, the server AP prepares and sends the next data block, including  $B_{k+1}, B_{k+2}, B_{k+3}, \dots, B_L$ , with block-number == 2. This exchange of sending data blocks and acknowledgements continues until the last data block, carrying  $B_M, B_{M+1}, B_{M+2}, \dots, B_N$  is sent. The last GET.response primitive is invoked with Response\_Type == LAST-BLOCK: in the DataBlock\_G structure Last\_Block == TRUE. This last data block is not acknowledged by the client.

Throughout the whole procedure, the Invoke\_Id and the Priority parameters shall be the same in each primitive.

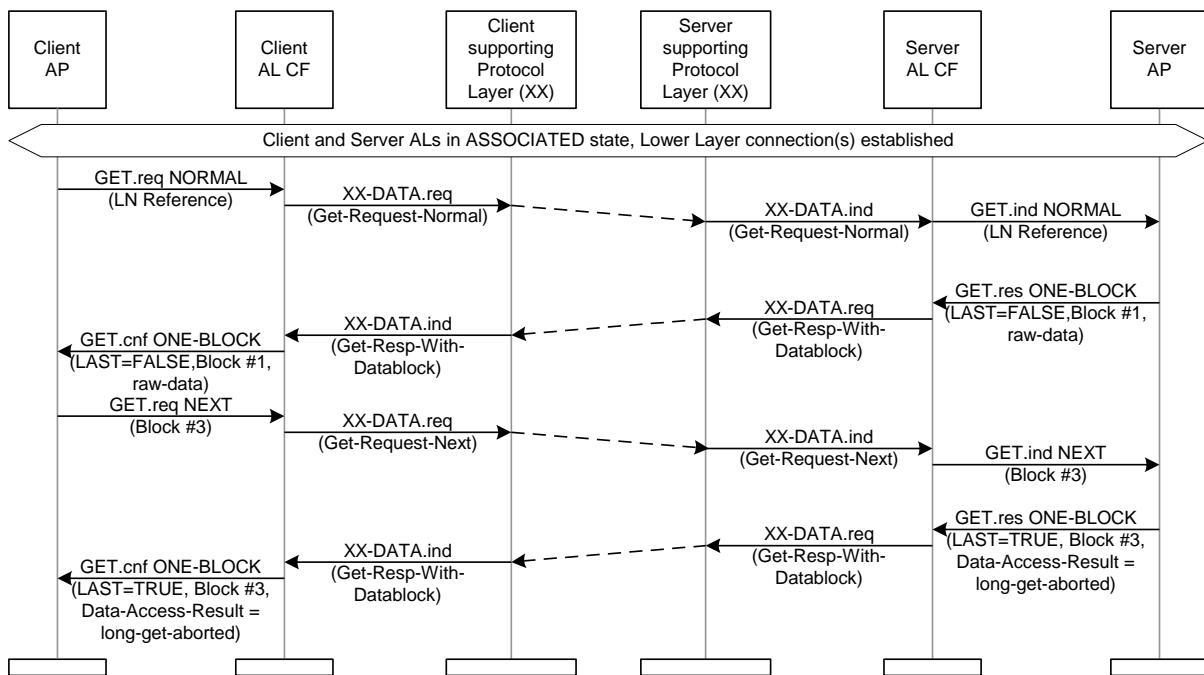
If during a long data transfer the server receives another service request, it is served according to the priority rules and the priority management settings (Conformance block bit 9).

If any error occurs during the long data transfer, the transfer is aborted. The error cases are:

- a) The server is not able to provide the next block of data for any reason. In this case, the server AP shall invoke a GET-RESPONSE-LAST-BLOCK service primitive. The Result parameter shall contain a DataBlock\_G structure with:
  - Last\_Block == TRUE;
  - Block\_Number == number of the block confirmed by the client +1;
  - Result == Data\_Access\_Result, indicating the reason of the failure.
- b) The Block\_Number parameter in a GET-REQUEST-NEXT service primitive is not equal to the number of the previous block sent by the server. The server interprets this, as if the client would like to abort the ongoing transfer. In this case, the server AP shall invoke a GET-RESPONSE-LAST-BLOCK service primitive. The Result parameter shall contain a DataBlock\_G structure with:
  - Last\_Block == TRUE;
  - Block\_Number == equal to the block number received from the client;
  - Result == Data-Access-Result, long-get-aborted.
- c) The server may receive a Get-Request-Next APDU when no long data transfer is in progress. In this case, the server AP shall invoke a GET-RESPONSE-LAST-BLOCK service primitive. The Result parameter shall contain a DataBlock\_G structure with:
  - Last-block == TRUE;
  - Block-Number == equal to the block number received from the client;
  - Result == Data-Access-Result, no-long-get-in-progress.
- d) The block number sent by the server is not equal to the next in sequence. In this case, the client shall abort the block transfer (see case b).

If, in the error cases above, the server is not able to invoke a GET-RESPONSE-LAST-BLOCK service primitive for any reason, it shall invoke a GET-RESPONSE-NORMAL service primitive, with the Data-Access-Result parameter indicating the reason of the failure. The server shall send a Get-Response-Normal APDU.

The MSC for error case b), long get aborted, is shown in Figure 122:

**Figure 122 – MSC of the GET service with block transfer, long GET aborted**

#### 9.4.6.4 Protocol for the SET service

When the client AP desires to write the value of one or more COSEM object attributes, it uses the SET service.

As explained in 9.3.7, the encoded form of the request may fit in a single request or not. In this latter case, either the service-specific or the general block transfer mechanism may be used. It is negotiated via bit 2 or bit 12 of the conformance block, see 9.4.6.1.

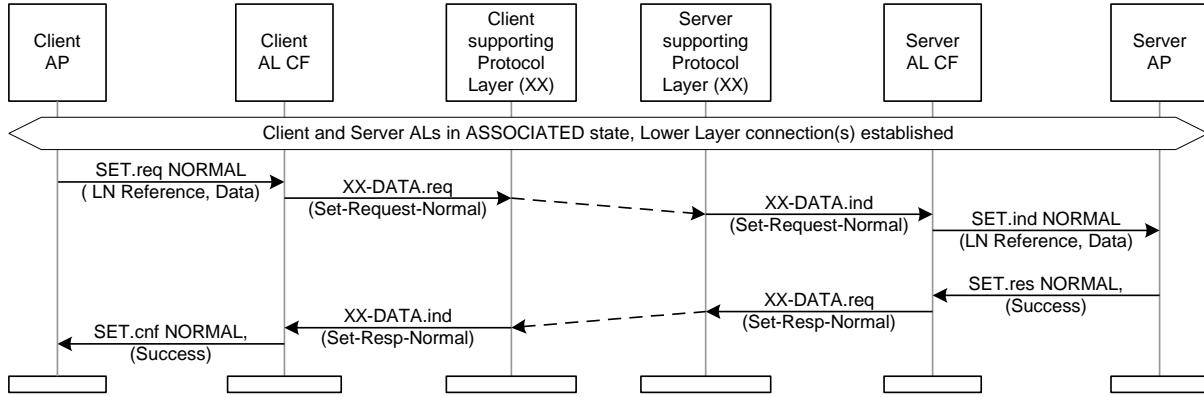
NOTE 1 In some DLMS/COSEM communication profiles segmentation is available to transfer long APDUs.

The SET service primitive types and the corresponding APDUs are shown in Table 87.

**Table 87 – SET service types and APDUs**

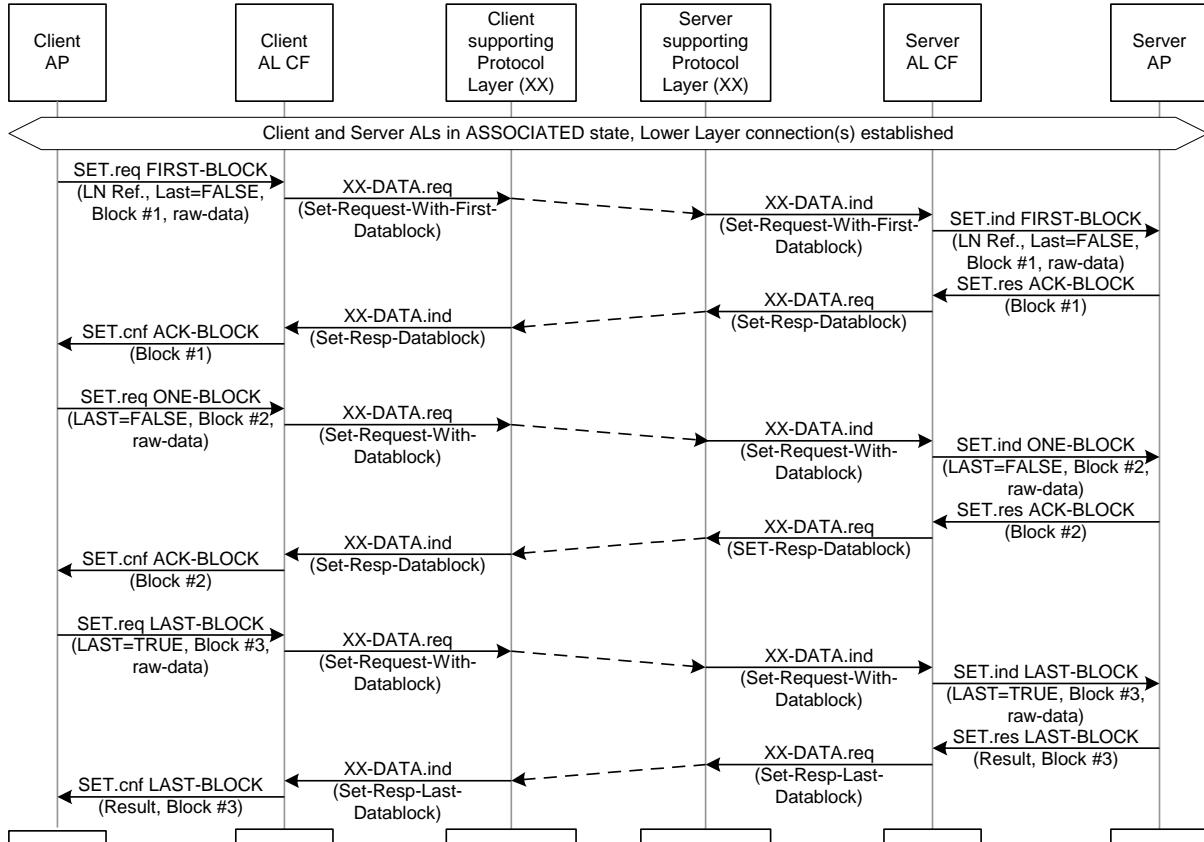
SET .req / .ind	Request APDU	Response APDU	SET .res / .cnf
NORMAL	Set-Request-Normal	Set-Response-Normal	NORMAL
FIRST-BLOCK	Set-Request-With-First-Datablock Last-Block = FALSE	Set-Response-Datablock	ACK-BLOCK
ONE-BLOCK	Set-Request-With-Datablock Last-Block = FALSE	Set-Response-Datablock	ACK-BLOCK
LAST-BLOCK	Set-Request-With-Datablock Last-Block = TRUE	Set-Response-Last-Datablock	LAST-BLOCK
		Set-Response-Last-Datablock-With-List	LAST-BLOCK-WITH-LIST
WITH-LIST	Set-Request-With-List	Set-Response-With-List	WITH-LIST
FIRST-BLOCK-WITH-LIST	Set-Request-With-List-And-With-First-Datablock	Set-Response-Datablock	ACK-BLOCK

Figure 123 shows the MSC of a confirmed SET service, in the case of success, without block transfer.



**Figure 123 – MSC of the SET service**

Figure 124 shows the MSC of a confirmed SET service in the case of success, with the request sent in three blocks, using the service-specific block transfer mechanism.



**Figure 124 – MSC of the SET service with block transfer**

As in this case the data to be sent is too long to fit in a single APDU, the client AP sends it in blocks. First, the data is encoded as if it would fit in a single APDU. The result is a series of bytes,  $B_1, B_2, B_3, \dots, B_N$ . The client may generate the complete request ( $B_1, B_2, B_3, \dots, B_N$ ) in one step or dynamically (on the fly).

The client AP assembles then a DataBlock\_SA structure:

- `Last_Block == FALSE;`

- Block\_Number == 1;
- Raw\_Data == the first K bytes of the encoded data: B<sub>1</sub>, B<sub>2</sub>, B<sub>3</sub>,..., B<sub>K</sub>.

The client AP invokes then the SET-REQUEST-FIRST-BLOCK or SET-REQUEST-FIRST-BLOCK-WITH-LIST service primitive as appropriate, carrying the attribute reference(s) and this DataBlock-SA structure.

Upon the reception of the .request primitive, the client AL builds the appropriate Set-Request APDU carrying the parameters of the .request primitive and sends it to the server.

The server stores the data block received, then acknowledges its reception and asks for the next one by invoking a SET.response primitive with Response\_Type == ACK-BLOCK and with the same block number as the number of the block received.

To send the next data block carrying B<sub>K+1</sub>, B<sub>K+2</sub>, B<sub>K+3</sub>,..., B<sub>L</sub>, the client AP invokes a SET-REQUEST-ONE-BLOCK service primitive. This exchange of sending data blocks and acknowledgements continues until the last data block carrying B<sub>M</sub>, B<sub>M+1</sub>, B<sub>M+2</sub>,..., B<sub>N</sub> is sent, by invoking a SET-REQUEST-LAST-BLOCK service primitive with Last\_Block == TRUE.

When these primitives are invoked, the client AL builds a Set-Request-With-Datablock APDU, carrying a DataBlock\_SA structure and sends these APDUs to the server.

When the server AP receives the last datablock, it invokes a SET-RESPONSE-LAST-BLOCK or SET-RESPONSE-LAST-BLOCK-WITH-LIST service primitive as appropriate. The Result parameter carries the result of the complete SET service invocation. The Block\_Number parameter confirms the reception of the last block.

Throughout the whole procedure, the Invoke\_Id and the Priority parameters shall be the same in each primitive.

If during a long data transfer the server receives another service request, it is served according to the priority rules and the priority management settings (Conformance block bit 9).

If any error occurs during the long data transfer, the transfer is aborted. The error cases are:

- a) The server is not able to handle the data block received, for any reason. In this case, the server AP shall invoke a SET-RESPONSE-LAST-BLOCK or SET-RESPONSE-LAST-BLOCK-WITH-LIST service primitive as appropriate. The Result parameter indicates the reason for aborting the transfer;
- b) The Block\_Number parameter in a SET-REQUEST-ONE-BLOCK service primitive is not equal to the block number expected by the server (last received + 1). The server interprets this as if the client would like to abort the ongoing transfer. In this case, the server AP shall invoke a SET-RESPONSE-LAST-BLOCK or SET-RESPONSE-LAST-BLOCK-WITH-LIST service primitive as appropriate with the Result parameter Data\_Access\_Result == long-set-aborted;
- c) The server may receive a Set-Request-With-Datablock APDU when no long data transfer is in progress. In this case, the server AP shall invoke a SET-RESPONSE-LAST-BLOCK service primitive with the Result parameter Data\_Access\_Result == no-long-set-in-progress.

If, in the error cases above, for any reason the server is not able to invoke a SET-RESPONSE-LAST-BLOCK service primitive, it invokes a SET-RESPONSE-NORMAL service primitive with the Data-Access-Result parameter indicating the reason of the failure.

#### 9.4.6.5 Protocol for the ACTION service

When the client AP desires to invoke one or more COSEM objects methods, it uses the ACTION service. As explained in 9.3.8, the ACTION service comprises two phases.

If the method references and method invocation parameters or the return parameters do not fit in a single APDU, either the service-specific or the general block transfer mechanism may be used. It is negotiated via bit 2 or bit 13 of the conformance block, see 9.4.6.1.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	329/614
-----------------------	------------	-----------------------	---------

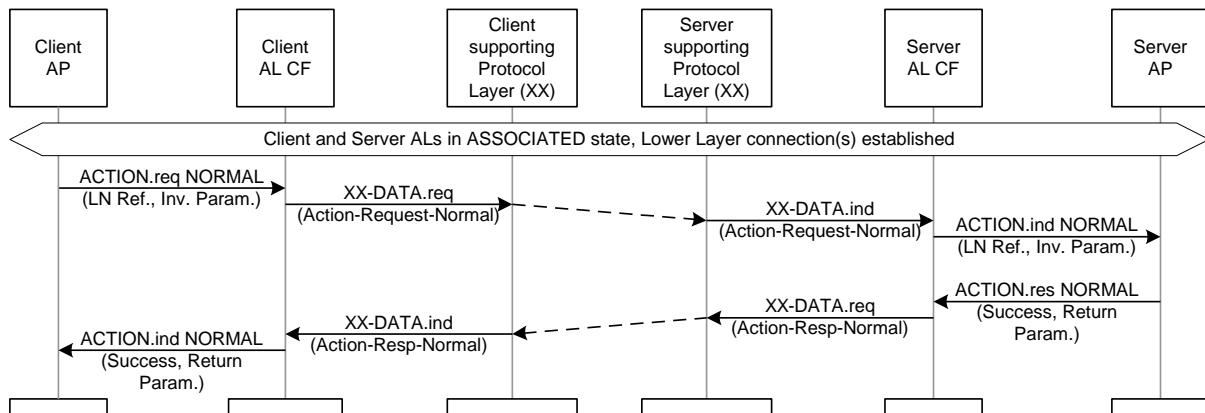
NOTE In some DLMS/COSEM communication profiles segmentation is available to transfer long APDUs.

The ACTION service primitive types and the corresponding APDUs are shown in Table 88.

**Table 88 – ACTION service types and APDUs**

ACTION.req / .ind	Request APDU	Response APDU	ACTION.res / .cnf
NORMAL	Action-Request-Normal	Action-Response-Normal	NORMAL
		Action-Response-With-Pblock	ONE-BLOCK
NEXT	Action-Request-Next-Pblock	Action-Response-With-Pblock	ONE-BLOCK
		Action-Response-With-Pblock	LAST-BLOCK
FIRST-BLOCK	Action-Request-With-First-Pblock	Action-Response-Next-Pblock	NEXT
ONE-BLOCK	Action-Request-With-Pblock		
LAST-BLOCK	Action-Request-With-Pblock	Action-Response-Normal	NORMAL
		Action-Response-With-Pblock	ONE-BLOCK
WITH-LIST	Action-Request-With-List	Action-Response-With-List	WITH-LIST
		Action-Response-With-Pblock	ONE-BLOCK
WITH-LIST-AND-FIRST-BLOCK	Action-Request-With-List-And-With-First-Pblock	Action-Response-Next-Pblock	NEXT

Figure 125 illustrates the MSC of a confirmed ACTION service in the case of success, without block transfer.



**Figure 125 – MSC of the ACTION service**

The ACTION service can transport data in both directions:

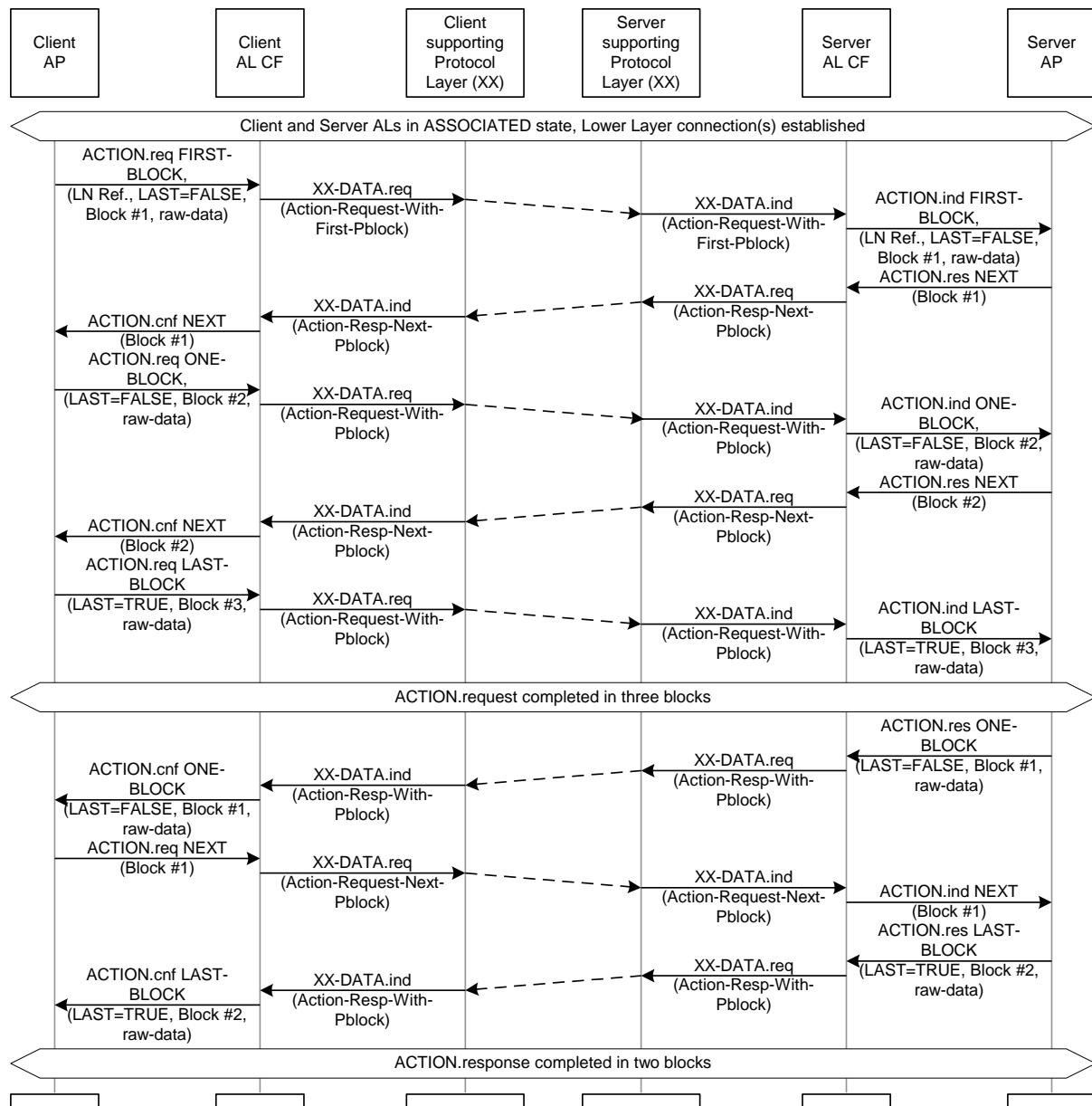
- in the first phase, the client sends the ACTION.request with the method invocation parameters for the method(s) referenced and the server acknowledges them. The process is essentially the same, as in the case of the SET service;
- in the second phase, the server sends the ACTION.response with the result of invoking the method(s) and the return parameters. The process is essentially the same as in the case of the GET service.

Throughout the whole procedure, the Invoke\_Id and the Priority parameters shall be the same in each primitive.

If during a long data transfer the server receives another service request, it is served according to the priority rules and the priority management settings (Conformance block bit 9).

Figure 126 illustrates the MSC in the case, when block transfer takes place in both directions using the service-specific block transfer mechanism.

If any error occurs during the long data transfer, the transfer shall be aborted. Error cases are the same as in the case of the GET and SET services.



**Figure 126 – MSC of the ACTION service with block transfer**

#### 9.4.6.6 Protocol for the ACCESS service

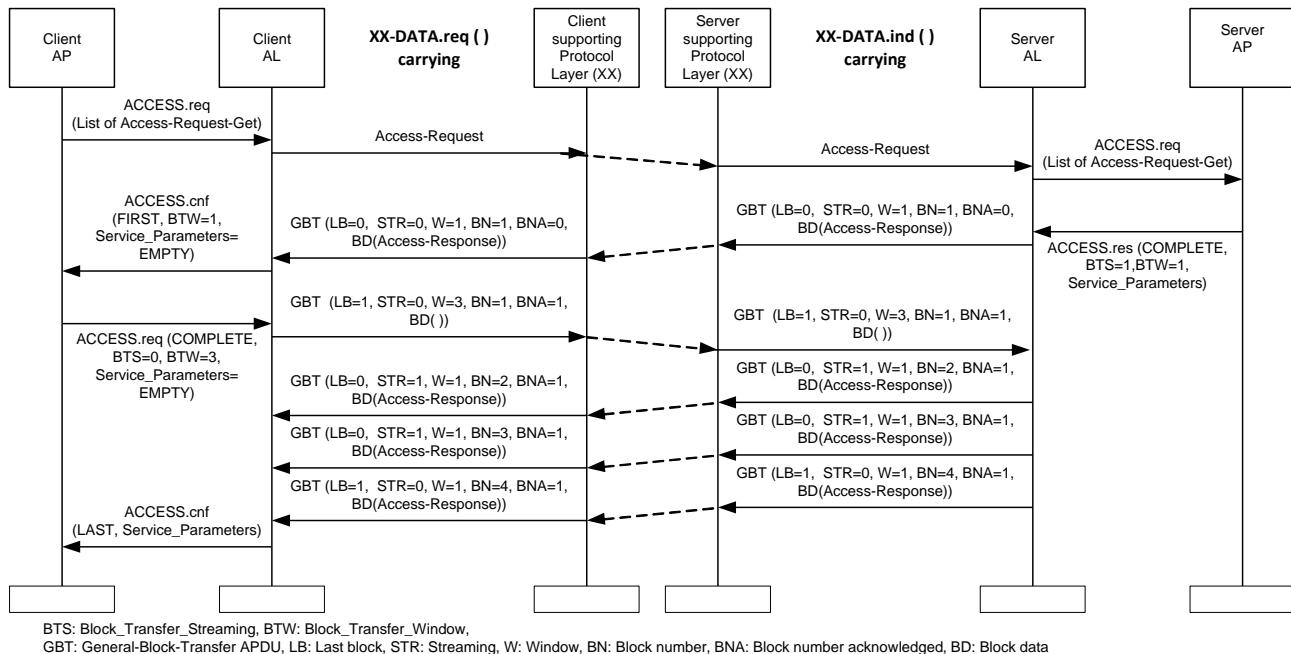
The client can use the ACCESS service to read or write the value of one or more COSEM object attributes or to invoke one or more methods.

The protocol of the ACCESS service is specified by way of message sequence charts, including cases where it is used together with general block transfer and general message protection.

NOTE See also 9.1.4.4.7, 9.3.5 and 9.4.6.13.

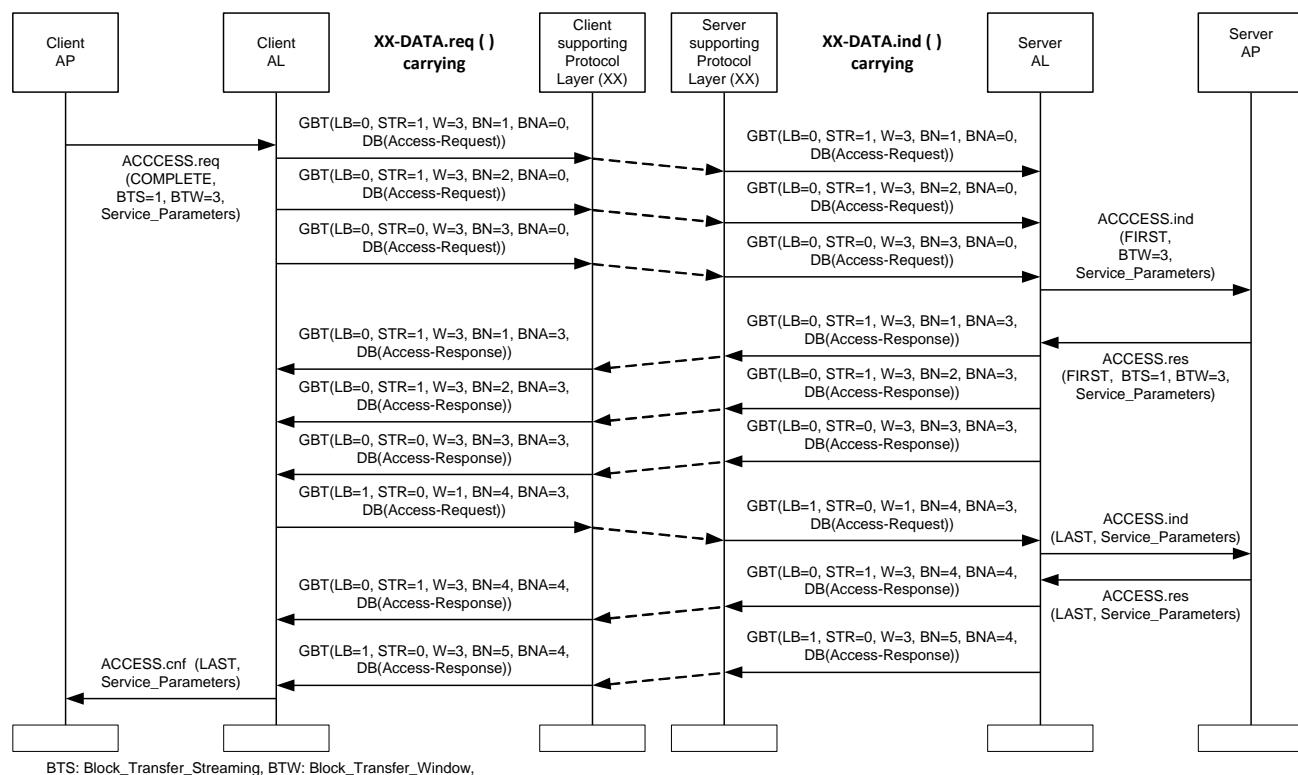
Figure 127 shows the MSC of an ACCESS service used to get one COSEM object attribute values. The request fits in a single APDU. The response is long therefore the server sends back the response using the GBT mechanism: portions of the access-response APDU are carried by the block-data field of general-block-transfer APDUs.

When the client receives the first general-block-transfer APDU, it switches to GBT announcing that streaming with window size 3 is supported.



**Figure 127 – ACCESS service with long response**

Figure 128 shows the MSC of an ACCESS service used to carry a list of requests, which does not fit in a single APDU, therefore GBT is used. The response is also long therefore the server also uses GBT. Both parties know a priori that the other party supports streaming with window size = 3.



**Figure 128 – ACCESS service with long request and response**

#### 9.4.6.7 Protocol of the DataNotification service

When the server AP invokes a DataNotification.request service primitive, the server AL builds the DataNotification APDU and passes it to the supporting protocol layer.

If the service primitives are long partial service invocations can be used.

If the encoded form of the service primitive is too long, then the general block transfer mechanism can be used. See also Figure 150.

When the client AL receives this APDU, it invokes the DataNotification.indication service primitive.

In case 1), unconfirmed, retry on supporting protocol layer failure (see 9.4.6.2.2):

- the push operation is deemed as failed when the server AP receives a DataNotification.cnf service primitive with Service\_Class == Unconfirmed and Result == LOWER\_LAYER\_FAILED. In this case, the server AP may attempt a retry after a repetition delay;
- the push operation is deemed as successful when the server AP receives a DataNotification.cnf with Service\_Class == Unconfirmed and Result == CONFIRMED;

Figure 129 shows the MSC of the DataNotification service for this case.

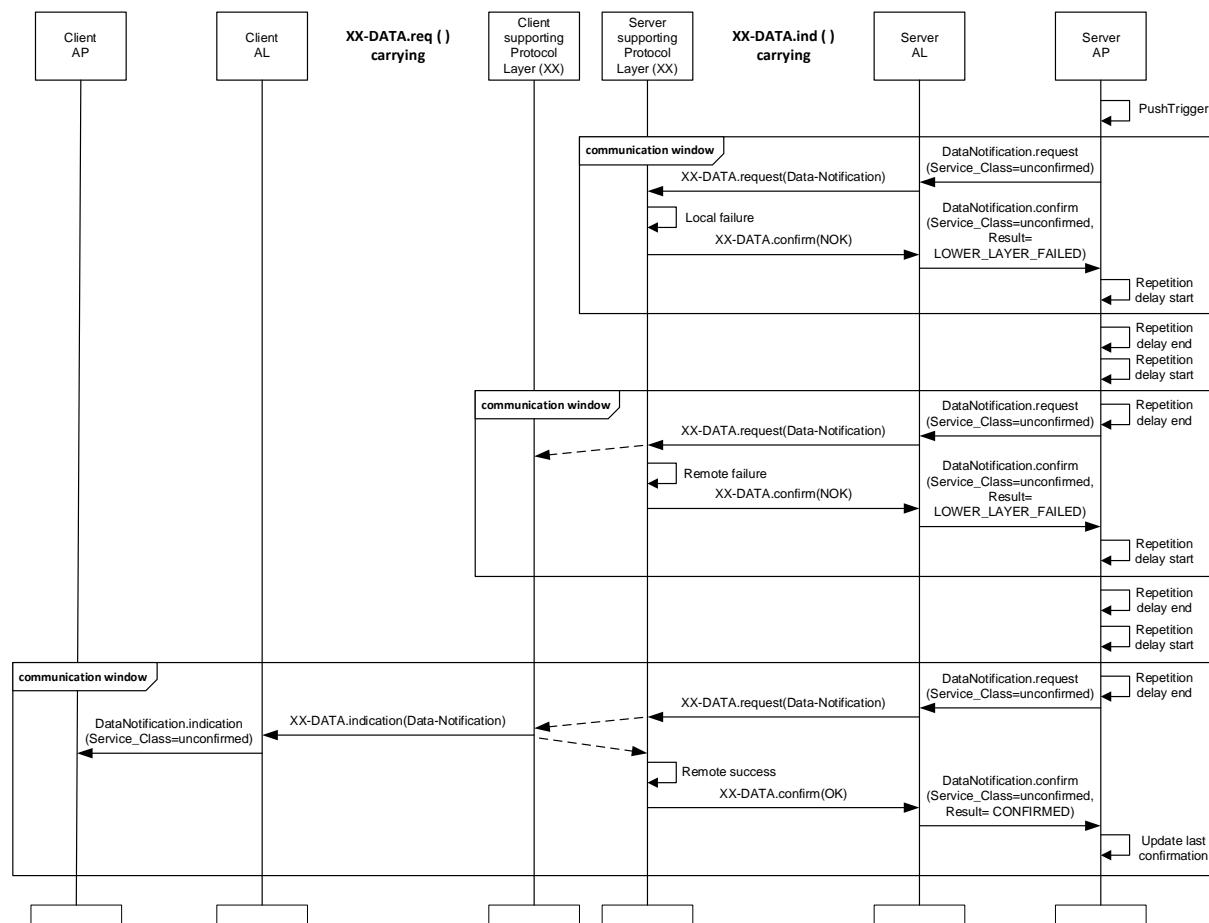


Figure 129 – MSC for the DataNotification service, case 1)

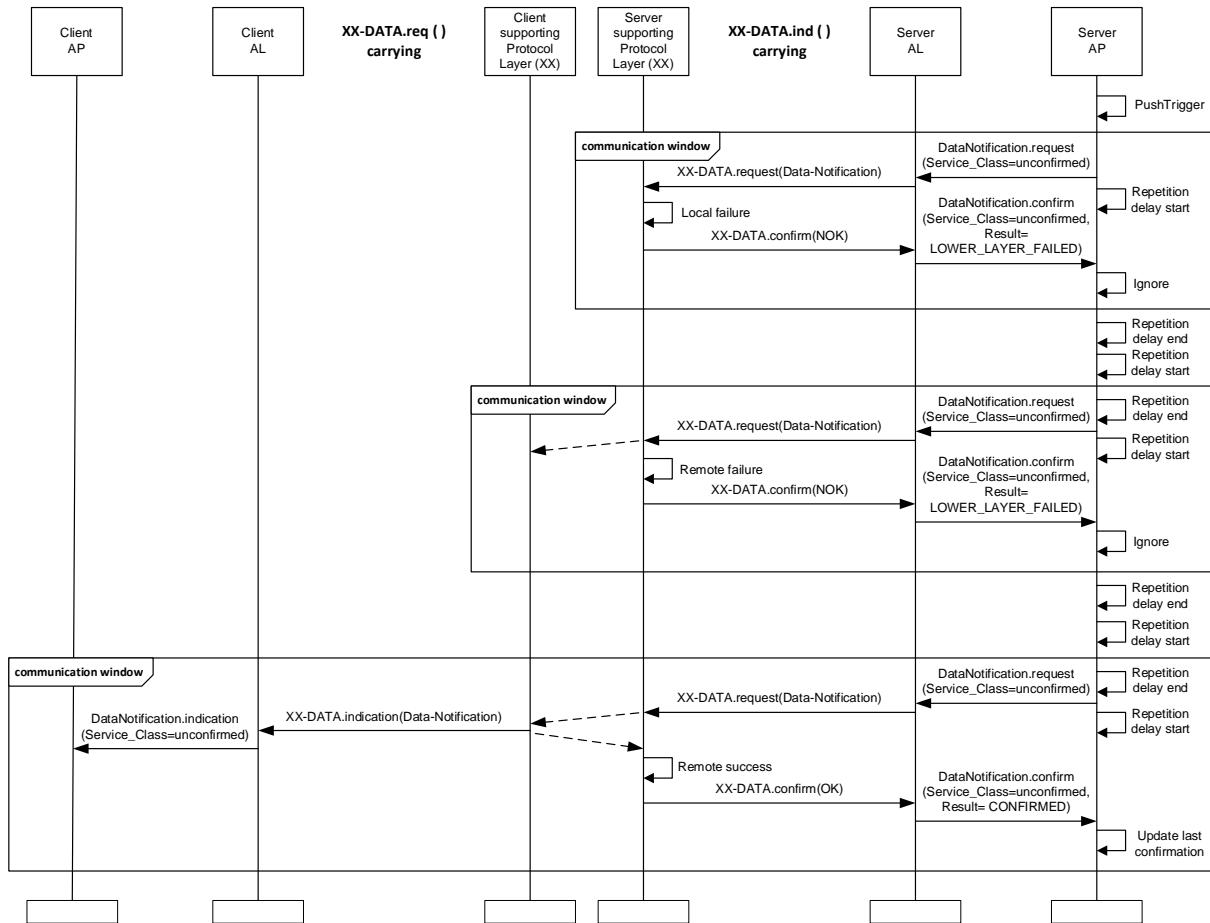
In case 2), unconfirmed, retry on missing supporting protocol layer confirmation (see 9.4.6.2.2):

- the push operation is deemed as failed if the repetition delay expires before a confirmation is received. In this case, the server may attempt a retry;
- the push operation is deemed as successful when the server supporting protocol layer informs the AL that the DataNotification APDU has been successfully received by the remote supporting

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	333/614
-----------------------	------------	-----------------------	---------

protocol layer. In this case, the server AL invokes a DataNotification.cnf service primitive with Service\_Class == Unconfirmed and Result == CONFIRMED.

Figure 130 shows the MSC of the DataNotification service for this case.



**Figure 130 – MSC for the DataNotification service, case 2)**

In case 3), confirmed, retry on missing confirmation (see 9.4.6.2.2):

- when the client AL receives a correct DataNotification APDU, it invokes the DataNotification.ind service primitive. If the Data-Notification APDU cannot be processed for any reason, it is discarded;
- the client AP invokes the DataNotification.response service primitive with Result == Confirmed. The client AL builds the Data-Notification-Confirm APDU and sends it to the server;
- the push operation is deemed as failed if the repetition delay expires before a confirmation is received. In this case, the server may attempt a retry;
- the push operation is deemed as successful when the AL layer receives the Data-Notification-Confirm APDU and invokes the DataNotification.cnf service primitive with Service\_Class == Confirmed and Result == CONFIRMED.

Figure 131 shows the MSC of the DataNotification service for this case.

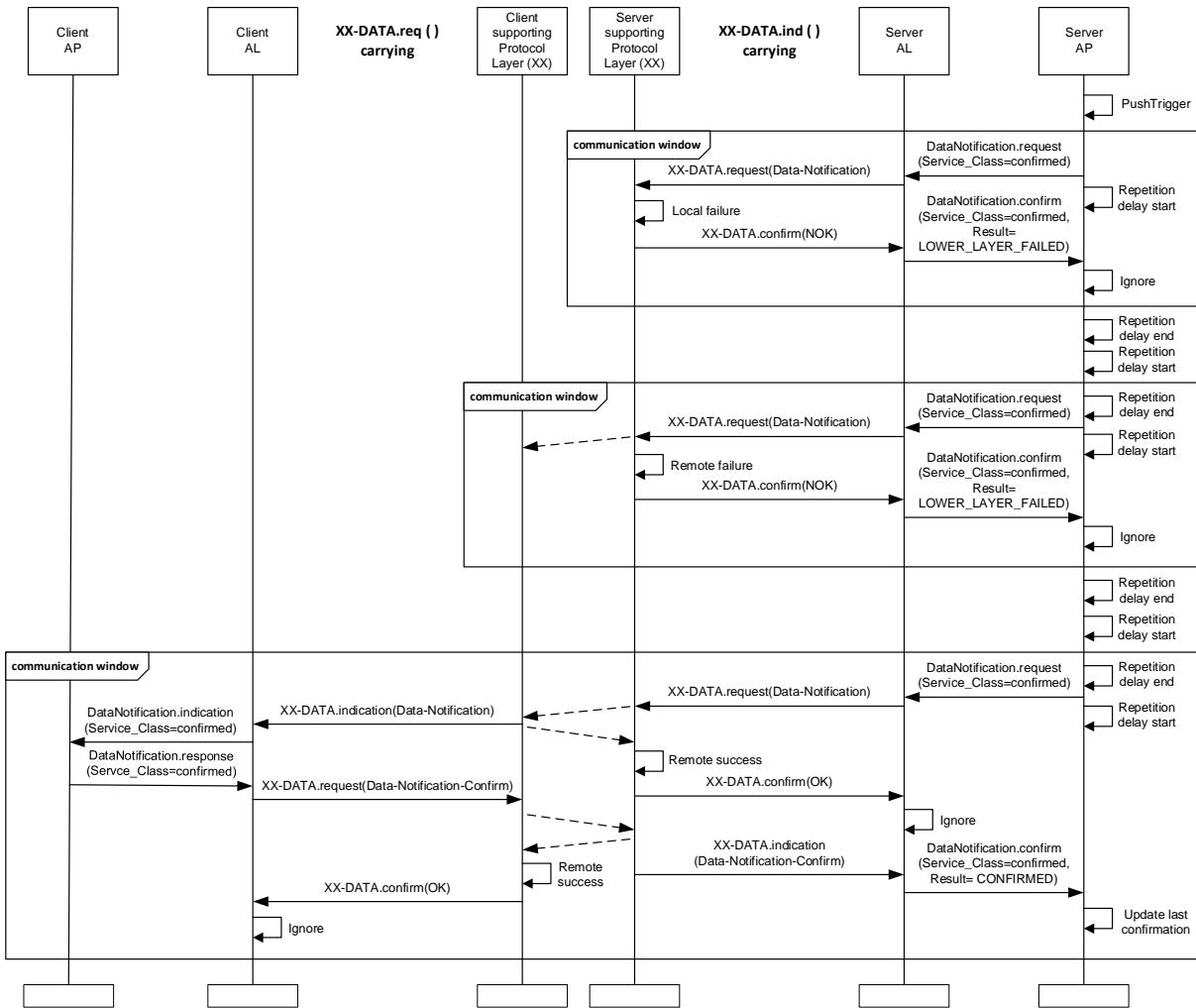


Figure 131 – MSC for the DataNotification service, case 3)

#### 9.4.6.8 Protocol for the EventNotification service

Upon invocation of the EventNotification.request service, the Server AL builds an event-notification-request APDU. The possibilities to send out this APDU depend on the communication profile and the connection status of the lower layers. Therefore, the protocol of the EventNotification service is further discussed in 10.

In any case, in order to send the value(s) of attribute(s) to the client, without the client requesting it:

- the server uses the EventNotification.request service primitive;
- upon the invocation of this primitive, the server AL builds an event-notification-request APDU;
- this APDU is carried by the supporting protocol layer service at the first opportunity to the client. The service type and the availability of this first opportunity depends on the communications profile used;
- upon the reception of the event-notification-request APDU, the client AL generates an EventNotification.indication primitive to the COSEM client AP;

NOTE At the client side, it is always EventNotification.indication, independently of the referencing scheme (LN or SN) used by the server.

- by default, event notifications are sent from the Management Logical Device (server) to the management AP (client).

#### 9.4.6.9 Protocol for the Read service

As explained in 9.3.14, the Read service is used when the server uses SN referencing, either to read (a) COSEM object attribute(s), or to invoke (a) method(s) when return parameters are expected:

- in the first case, the GET.request service primitives are mapped to Read.request primitives and the Read.confirm primitives are mapped to GET.confirm primitives. The mapping and the corresponding SN APDUs is shown in Table 89;
- in the second case, the ACTION.request service primitives are mapped to Read.request primitives and the Read.response primitives are mapped to ACTION.response primitives. The mapping and the corresponding SN APDUs is shown in Table 90.

NOTE In the mapping tables below, the following notation is used:

- for LN services, only the request and response types are shown without service parameters;
- for SN services, the name of the service primitive is followed by the service parameters in brackets. Service parameter name elements are capitalized and joined with an underscore to signify a single entity. Parameters that may be repeated are shown in curly brackets. The choices that can be taken for the Variable\_Access\_Specification parameter are listed following the symbol “=”. Alternatives are separated by the vertical bar “|”.
- for SN APDUs, the name of the APDU is followed by the symbol “::=” and the fields in brackets. The field name elements are not capitalized and are joined with a dash to signify a single entity. Fields that may be repeated are shown in curly brackets. Alternatives are separated by the vertical bar “|”.

**Table 89 – Mapping between the GET and the Read service**

<b>From GET.request of type</b>	<b>To Read.request</b>	<b>SN APDU</b>
NORMAL	Read.request (Variable_Access_Specification) Variable_Access_Specification = Variable_Name   Parameterized_Access;	ReadRequest ::= (variable-name   parameterized-access)
NEXT	Read.request (Variable_Access_Specification) Variable_Access_Specification = Block_Number_Access;	ReadRequest ::= (block-number-access)
WITH-LIST	Read.request ({Variable_Access_Specification}) Variable_Access_Specification = Variable_Name   Parameterized_Access;	ReadRequest ::= ({variable-name   parameterized-access})
<b>To GET.confirm of type</b>	<b>SN APDU</b>	<b>From Read.response</b>
NORMAL	ReadResponse ::= (data   data-access-error)	Read.response (Data   Data_Access_Error)
ONE-BLOCK	ReadResponse ::= (data-block-result)	Read.response (Data_Block_Result) with Last_Block = FALSE
LAST-BLOCK	ReadResponse ::= (data-block-result)	Read.response (Data_Block_Result) with Last_Block = TRUE
WITH-LIST	ReadResponse ::= ({data   data-access-error})	Read.response ({Data   Data_Access_Error})

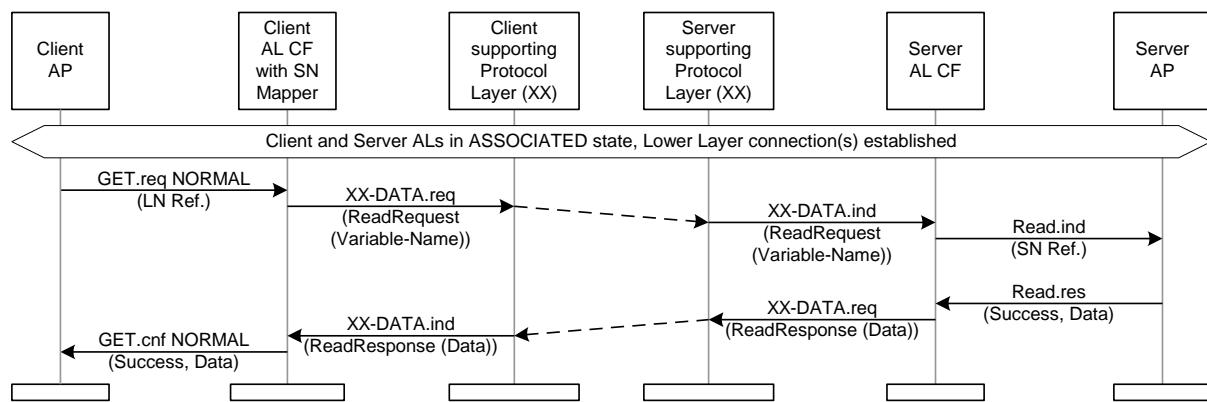
**Table 90 – Mapping between the ACTION and the Read service**

<b>From ACTION.request of type</b>	<b>To Read.request</b>	<b>SN APDU</b>
NORMAL	Read.request (Variable_Access_Specification) Variable_Access_Specification = Parameterized_Access; with Variable_Name = method reference, Selector = 0, Parameter = method invocation parameter or null-data	ReadRequest ::= (parameterized-access)
NEXT	Read.request (Variable_Access_Specification) Variable_Access_Specification = Block_Number_Access;	ReadRequest ::= (block-number-access)
FIRST-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1, Raw_Data = one part of the method reference(s) and method invocation parameter	ReadRequest ::= (read-data-block-access)

**Table 82 Continued.**

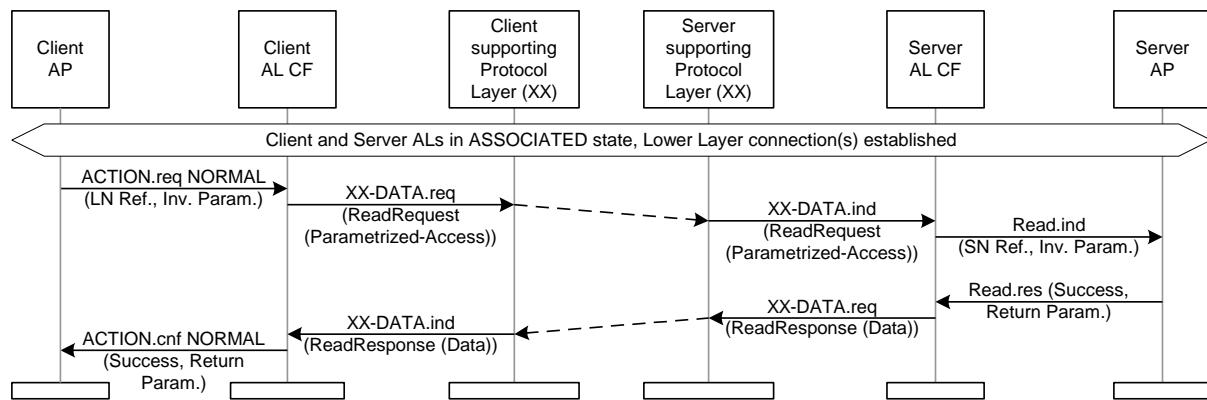
<b>From ACTION.request of type</b>	<b>To Read.request</b>	<b>SN APDU</b>
ONE-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; with Last_Block = FALSE, Block_Number = next number, Raw_Data = as above	ReadRequest ::= (read-data-block-access)
LAST-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; with Last_Block = TRUE, Block_Number = next number, Raw_Data = as above	ReadRequest ::= (read-data-block-access)
WITH-LIST	Read.request ({Variable_Access_Specification}) Variable_Access_Specification = Parameterized_Access; with Variable_Name = method reference, Selector = 0, Parameter = method invocation parameter or null-data	ReadRequest ::= (parameterized-access)
WITH-LIST-AND-FIRST-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1, Raw_Data = as above	ReadRequest ::= (read-data-block-access)
<b>To ACTION.confirm</b>	<b>SN APDU</b>	<b>From Read.response</b>
NORMAL	ReadResponse ::= (data   data-access-error)	Read.response (Read_Result) Read_Result = Data   Data_Access_Error;
ONE-BLOCK	ReadResponse ::= (data-block-result)	Read.response (Read_Result) Read_Result = Data_Block_Result; with Last_Block = FALSE
LAST-BLOCK	ReadResponse ::= (data-block-result)	Read.response (Read_Result) Read_result = Data_Block_Result; with Last_Block = TRUE
NEXT	ReadResponse ::= (block-number)	Read.confirm (Read_Result) Read_Result = Block_Number;
WITH-LIST	ReadResponse ::= ({data   data-access-error})	Read.response ({Read_Result}) Read_Result = Data   Data_Access_Error;

Figure 132 shows the MSC of a Read service used to read the value of a single attribute.



**Figure 132 – MSC of the Read service used for reading an attribute**

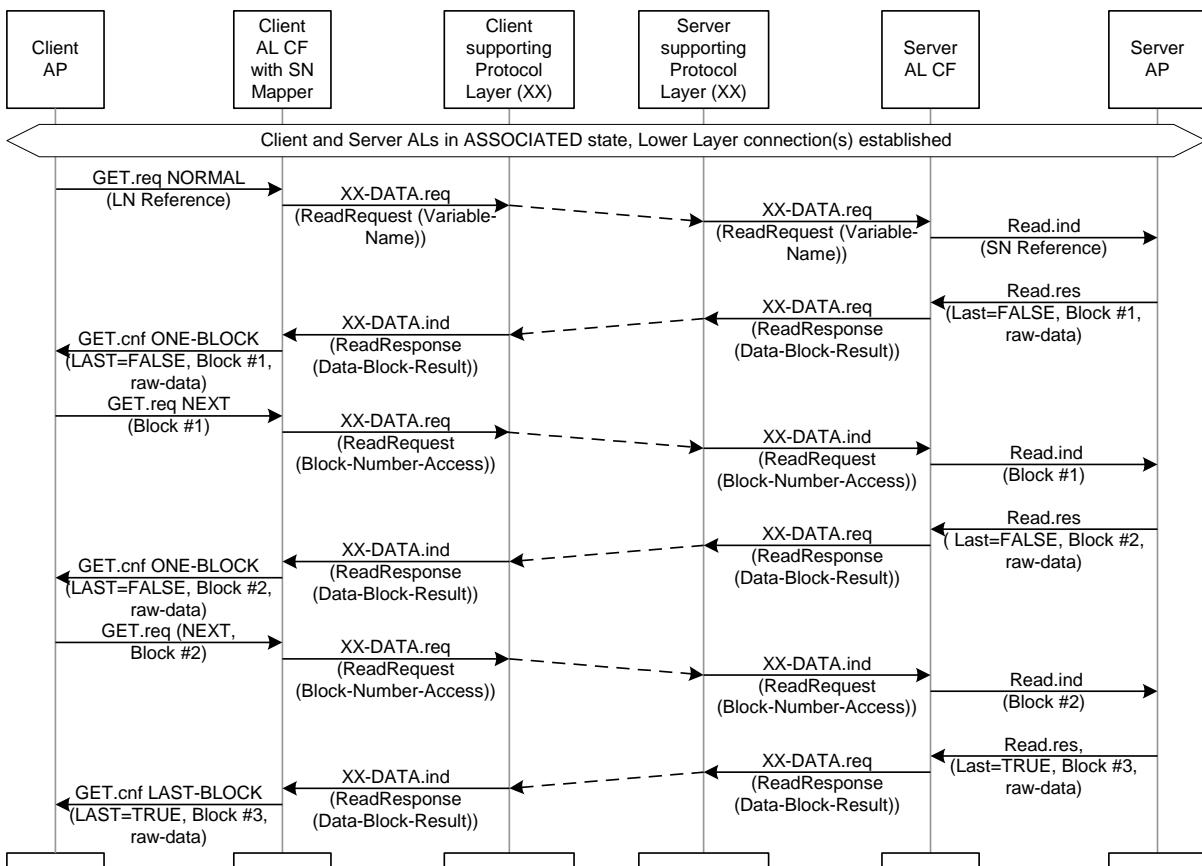
Figure 133 shows the MSC of a Read service used to invoke a single method.



**Figure 133 – MSC of the Read service used for invoking a method**

Figure 134 shows the MSC of a Read service for reading a single attribute, with the result returned in three blocks using the service-specific block transfer mechanism.

Alternatively, the general block transfer mechanism can be used.

**Figure 134 – MSC of the Read service used for reading an attribute, with block transfer**

The process of preparing and transporting the long data is essentially the same as in the case of the GET and ACTION service.

- if the Read service is used to read the value of (a) COSEM object attribute(s), the Raw\_Data element of the Data\_Block\_Result construct carries one part of the list of Read\_Result(s);
- if the Read service is used to invoke (a) COSEM object method(s) and long method invocation parameters have to be sent, the Raw\_Data element of the Read\_Data\_Block\_Access construct carries one part of the method reference(s) and method invocation parameter(s). If long method invocation responses are returned, the Raw\_Data element of the Data\_Block\_Result construct carries one part of the method invocation response(s).

If an error occurs, the server should return a Read.response service primitive with Data\_Access\_Error carrying appropriate diagnostic information; for example data-block-number-invalid.

#### 9.4.6.10 Protocol for the Write service

As explained in 9.3.15, the Write service is used when the server uses SN referencing, either to write (a) COSEM object attribute(s), or to invoke (a) method(s) when no return parameters are expected:

- in the first case, the SET.request service primitives are mapped to Write.request primitives and the Write.confirm primitives to SET.confirm primitives. The mapping and the corresponding SN APDUs are shown in Table 91;
- in the second case, the ACTION.request service primitives are mapped to Write.request primitives and the Write.response primitives to ACTION.confirm primitives. The mapping and the corresponding SN APDUs are shown in Table 92.

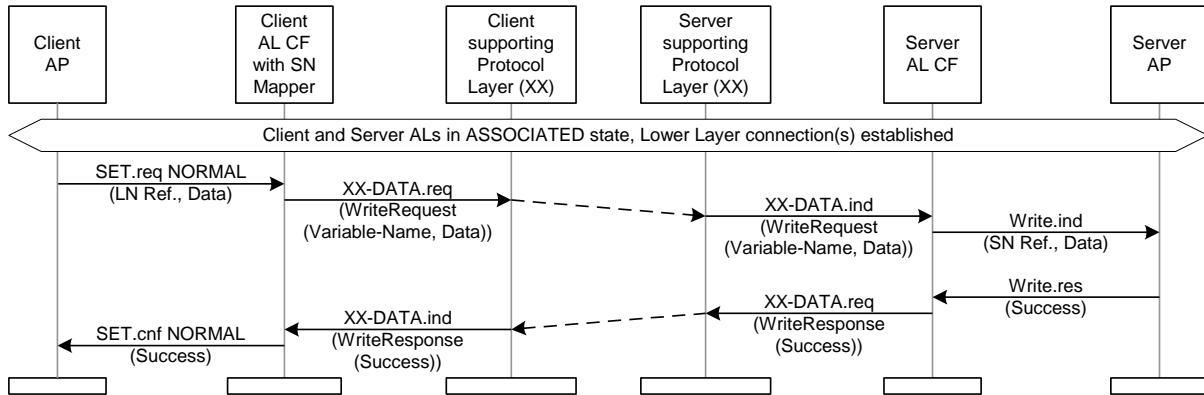
**Table 91 – Mapping between the SET and the Write service**

<b>From SET.request of type</b>	<b>To Write.request</b>	<b>SN APDU</b>
NORMAL	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name I Parameterized_Access;	WriteRequest ::= (variable-name I parameterized-access, data)
FIRST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1, Data = raw-data, carrying the encoded form of the attribute reference(s) and write data.	WriteRequest ::= (write-data-block-access, data)
ONE-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = next number, Data = as above	WriteRequest ::= (write-data-block-access, data)
LAST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = TRUE, Block_Number = next number, Data = as above	WriteRequest ::= (write-data-block-access, data)
WITH-LIST	Write.request ({Variable_Access_Specification}, {Data}) Variable_Access_Specification = Variable_Name I Parameterized_Access;	WriteRequest ::= ({variable-name I parameterized-access}, {data})
FIRST-BLOCK-WITH-LIST	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1 Data = as above	WriteRequest ::= (write-data-block-access, data)
<b>To SET.confirm of type</b>	<b>SN APDU</b>	<b>From Write.response</b>
NORMAL	WriteResponse ::= (success I data-access-error)	Write.response (Write_Result) Write_Result = Success I Data_Access_Error;
ACK-BLOCK	WriteResponse ::= (block-number)	Write.response (Write_Result) Write_Result = Block_Number;
LAST-BLOCK	WriteResponse ::= (success I data-access-error)	Write.response (Write_Result) Write_Result = Success I Data_Access_Error;
WITH-LIST	WriteResponse ::= ({success I data-access-error})	Write.response ({Write_Result}) Write_Result = Success I Data_Access_Error;
LAST-BLOCK-WITH-LIST	WriteResponse ::= ({success I data-access-error})	Write.response ({Write_Result}) Write_Result = Success I Data_Access_Error;

**Table 92 – Mapping between the ACTION and the Write service**

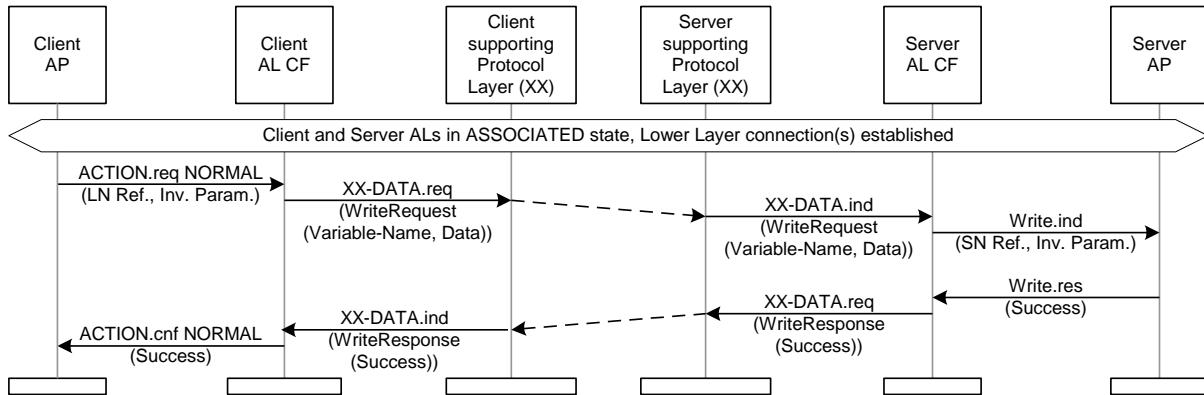
<b>From ACTION.request of type</b>	<b>To Write.request</b>	<b>SN APDU</b>
NORMAL	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name; Data = method invocation parameters or null-data	WriteRequest ::= (variable-name, data)
FIRST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1,  Data = raw-data, carrying the encoded form of the method reference(s) and method invocation parameters;	WriteRequest ::= (write-data-block-access, data)
ONE-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = next number,  Data = as above	WriteRequest ::= (write-data-block-access, data)
LAST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = TRUE, Block_Number = next number,  Data = as above	WriteRequest ::= (write-data-block-access, data)
WITH-LIST	Write.request ({Variable_Access_Specification}, {Data}) Variable_Access_Specification = Variable_Name; Data = method invocation parameters or null data	WriteRequest ::= ({variable-name}, {data})
WITH-LIST-AND-FIRST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1,  Data = as with first block	WriteRequest ::= (write-data-block-access, data)
<b>To ACTION.confirm</b>	<b>SN APDU</b>	<b>From Write.response</b>
NORMAL	WriteResponse ::= (success   data-access-error)	Write.response (Write_Result) Write_Result = Success   Data_Access_Error
NEXT	WriteResponse ::= (block-number)	Write.response (Block_Number)
<b>To ACTION.confirm</b>	<b>SN APDU</b>	<b>From Write.response</b>
WITH-LIST	WriteResponse ::= ({success   data-access-error})	Write.response ({Write_Result}) Write_Result = Success   Data_Access_Error

Figure 135 shows the MSC of a Write service used to write the value of a single attribute, in the case of success.



**Figure 135 – MSC of the Write service used for writing an attribute**

Figure 136 shows the MSC of a Write service used to invoke a single method, in the case of success.



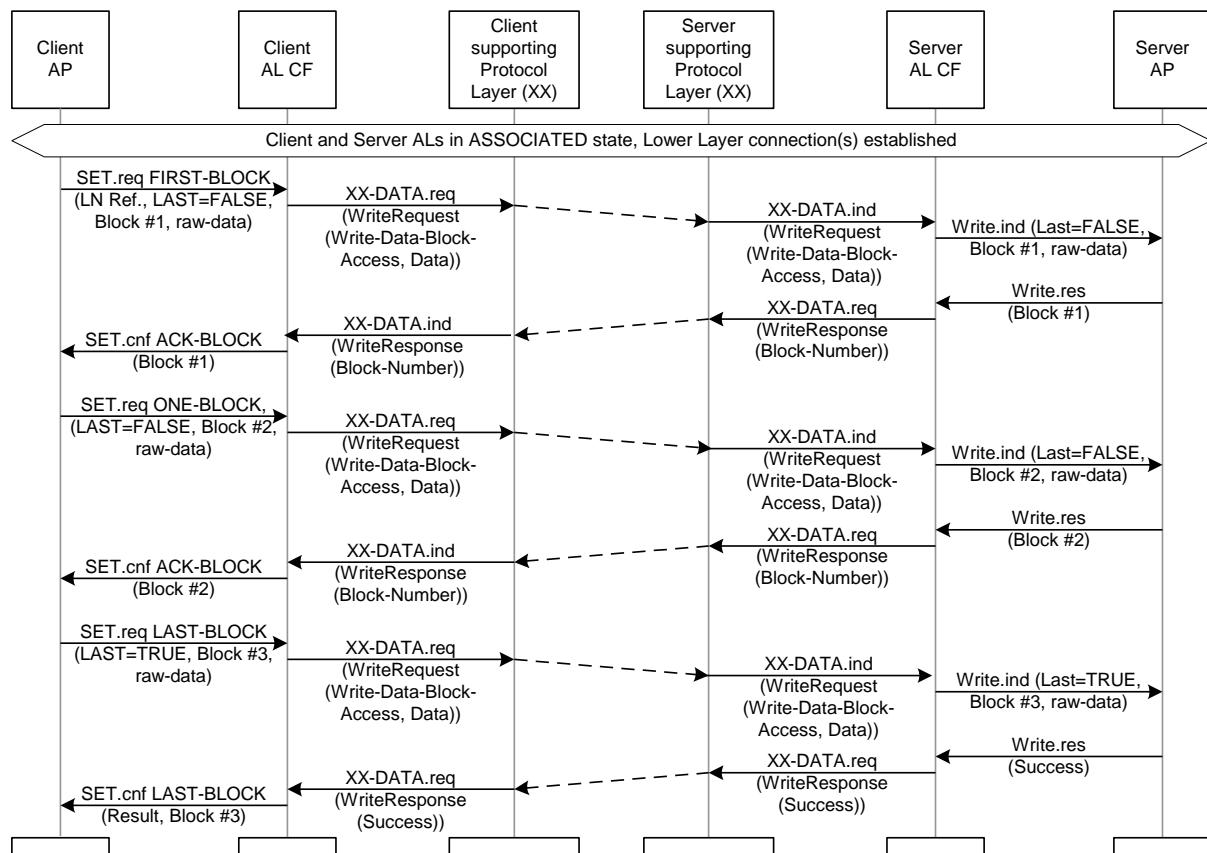
**Figure 136 – MSC of the Write service used for invoking a method**

Figure 137 shows the MSC of a Write service for writing a single attribute with the result returned in three blocks using the service-specific block transfer mechanism.

Alternatively, the general block transfer mechanism can be used.

The process of preparing and transporting the long data is essentially the same as in the case of the SET and ACTION service.

If an error occurs, the server should return a Write.response service primitive with the Data\_Access\_Error carrying appropriate diagnostic information for example data-block-number-invalid.

**Figure 137 – MSC of the Write service used for writing an attribute, with block transfer**

#### 9.4.6.11 Protocol for the UnconfirmedWrite service

This service may be invoked only when an AA has already been established. Depending on the communication profile, the APDU corresponding to the request may be transported using the connection-oriented or connectionless data services of the supporting protocol layer.

As explained in 9.3.16, the UnconfirmedWrite service may be used either to write (a) COSEM object attribute(s), or to invoke (a) method(s) when no return parameters are expected:

- in the first case, the SET.request service primitives are mapped to UnconfirmedWrite.request primitives. The mapping and the corresponding SN APDUs are shown in Table 93;
- in the second case, the ACTION.request service primitives are mapped to UnconfirmedWrite.request primitives. The mapping and the corresponding SN APDUs are shown in Table 94.

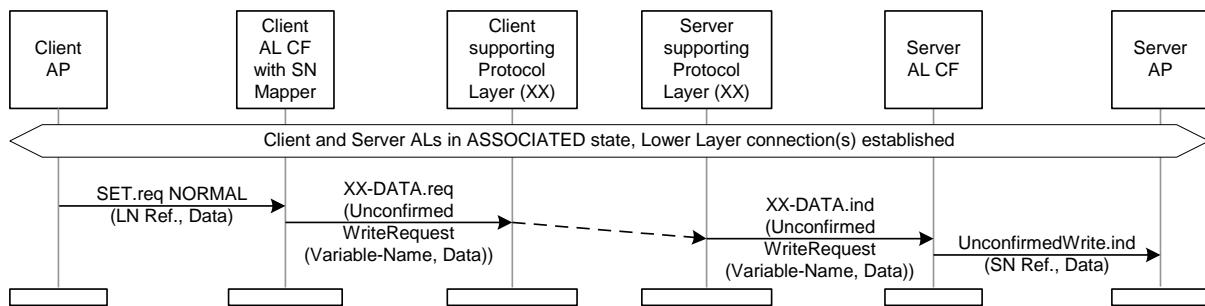
**Table 93 – Mapping between the SET and the UnconfirmedWrite service**

From SET.request of type	To UnconfirmedWrite.request	SN APDU
NORMAL	UnconfirmedWrite.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name   Parameterized_Access;	UnconfirmedWriteRequest ::= (variable-name   parameterized-access, data)
WITH-LIST	UnconfirmedWrite.request ({Variable_Access_Specification}, {Data}) Variable_Access_Specification = Variable_Name   Parameterized_Access;	UnconfirmedWriteRequest ::= ({variable-name   parameterized-access}, {data})

**Table 94 – Mapping between the ACTION and the UnconfirmedWrite service**

From ACTION.request of type	To UnconfirmedWrite.request	SN APDU
NORMAL	UnconfirmedWrite.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name; Data = method invocation parameters or null data	UnconfirmedWriteRequest ::= (variable-name, data)
WITH-LIST	UnconfirmedWrite.request ({Variable_Access_Specification}, {Data}) Variable_Access_Specification = Variable_Name; Data = as above	UnconfirmedWriteRequest ::= ({variable-name}, {data})

Figure 138 shows the MSC of a Write service used to write the value of a single attribute, in the case of success.

**Figure 138 – MSC of the UnconfirmedWrite service used for writing an attribute**

When the service parameters are long, the general block transfer mechanism can be used.

#### 9.4.6.12 Protocol for the InformationReport service

The protocol for the InformationReport service, specified 9.3.17 in is essentially the same as that of the EventNotification service, 9.4.6.6.

As, unlike the EventNotification service, the InformationReport service does not contain the optional Application\_Addresses parameter, the information report is always sent by the Server Management Logical Device to the Client Management AP.

Upon invocation of the InformationReport.request service, the server AP builds an informationReportRequest APDU. This APDU is sent from the SAP of the Management Logical Device to the SAP of the Client Management AP, using data services of the lower layers, in a non-solicited manner, at the first available opportunity.

The possibilities to send out this APDU depend on the communication profile and the connection status of the lower layers. Therefore, the protocol of the InformationReport service is further discussed in 10.

The InformationReport service may carry several attribute names and their contents. On the other hand, the EventNotification service specified in 9.3.9 contains only one attribute reference. Therefore, when the informationReportRequest APDU contains more than one attribute, it must be mapped to several EventNotification.ind services, as shown in Table 95.

**Table 95 – Mapping between the EventNotification and InformationReport services**

<b>EventNotification.ind (one or more)</b>	<b>InformationReport.ind</b>
Time (optional)	Current-time (optional)
COSEM_Class_Id, COSEM_Object_Instance_Id, COSEM_Object_Attribute_Id	Variable_Name {Variable_Name}
Attribute_Value	Data {Data}

#### 9.4.6.13 Protocol of general block transfer mechanism

##### 9.4.6.13.1 Introduction

The general block transfer (GBT) mechanism can be used to carry any xDLMS APDU when the service parameters are long i.e. their encoded form with the protection overhead, if any, is longer than the Max Receive PDU Size of the peer negotiated. In this case, the AL uses one or more General-Block-Transfer (GBT) xDLMS APDUs to transport such long APDUs.

The service primitive invocations may be complete including all the service parameters, or partial including only one part of the service parameters. Using complete or partial service invocations is left to the implementation.

Following the reception of a service .request / .response service primitive from the AP, the AL:

- builds the APDU that carries the service primitive;
- when ciphering is required it applies the protection as required by the Security\_Options and builds the appropriate ciphered APDU;
- when the resulting APDU is longer than the negotiated max APDU size, then the AL uses the GBT mechanism to send the complete message in several GBT APDUs.

However, there is no direct relationship between partial invocations and the GBT APDUs sent. The AL may apply the protection using complete or partial service invocations.

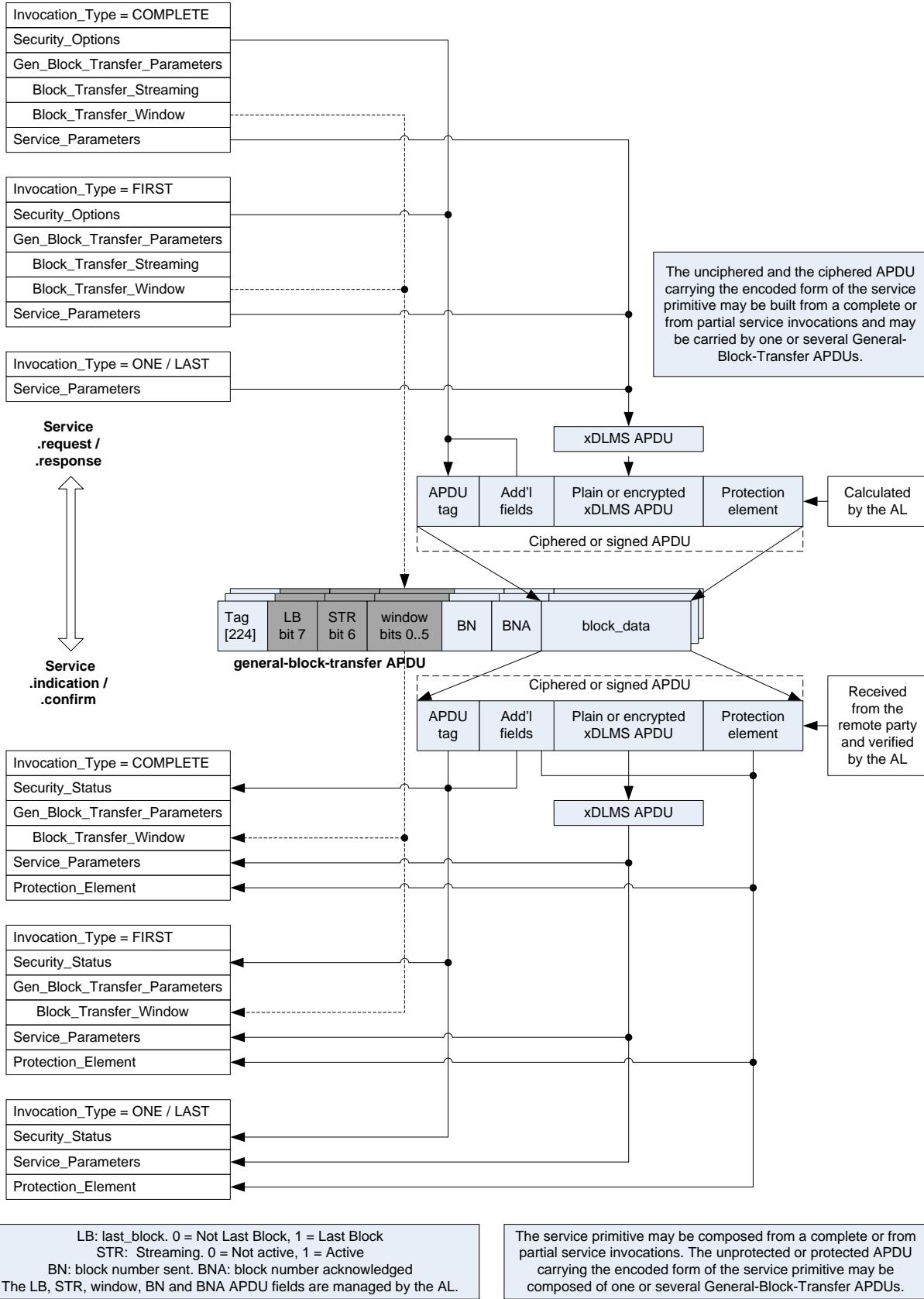
Following the reception of GBT APDUs from a remote party, the AL:

- assembles the block-data fields of the GBT APDUs received together;
- when the resulting complete APDU is ciphered, it checks and removes the protection;
- it invokes the appropriate service primitive, passing the additional Security\_Status, the General\_Block\_Transfer\_Parameters and the Protection\_Element.

However, there is no direct relationship between the GBT APDUs received and the partial service invocations. The AL may verify and remove the protection processing the GBT APDUs or processing the complete, assembled APDU.

See also Figure 113.

A confirmed message exchange may be started without or with using GBT. However, if one party sends a request or a response using GBT, the other party shall follow. The parties continue then using GBT until the end, i.e. until the complete response will have been received.



NOTE Applying and checking/removing cryptographic protection on APDUs is independent from the GBT process. It is included here for completeness.

**Figure 139 – Partial service invocations and GBT APDUs**

Streaming of blocks is managed by the AL taking into account the GBT parameters passed from the local AP to the AL – see 9.3.5 – and the fields of GBT APDUs – see Figure 139 – received from the remote AL.

The various service invocation types – COMPLETE, FIRST-PART, ONE-PART or LAST-PART – and the relationship between these invocations, the service parameters and the fields of the ciphered APDUs and the General-Block-Transfer (GBT) APDUs are shown in Figure 139.

The Block\_Transfer\_Streaming (BTS) parameter is passed by the AP to the AL to indicate that the AL can send blocks in streams, i.e. without waiting for a confirmation of each block received by the remote party. This parameter is not included in the APDU.

The Block\_Transfer\_Window (BTW) parameter indicates the size of the streaming window supported, i.e. the maximum number of blocks that can be received. The Block\_Transfer\_Window parameter of the other party may be known *a priori* by the parties. However, the window size is managed by the AL: it can use a lower value, for example during lost block recovery.

NOTE 1 This relationship is indicated using a dotted line in Figure 139 between the Block\_Transfer\_Window parameter and the window field of the APDU.

In the case of unconfirmed services the Block\_Transfer\_Streaming parameter shall be set to FALSE and the Block\_Transfer\_Window shall be set to 0. This indicates to the AL that it shall send the encoded form of the whole service primitive in as many GBT APDUs as needed without waiting for confirmation of the blocks sent.

The use of the fields of the GBT APDU is specified below:

- the last-block (LB) bit indicates if the block is the last one (LB = 1) or not (LB = 0). For the management of the last-block field see 9.4.6.13.4.3.2;
- the streaming bit indicates if streaming is in progress (STR = 1) or finished (STR = 0). When streaming is finished, the remote party shall confirm the blocks received. However, the client does not confirm the reception of the blocks received in the last stream. Blocks received in an unconfirmed GBT are not confirmed either. When the Block\_Transfer\_Streaming parameter has been set to FALSE, the streaming bit shall be also set to 0;
- the window field indicates the number of blocks that can be received by the party sending the GBT APDU. Its maximum value is equal to the Block\_Transfer\_Window parameter passed by the AP to the AL. Note, that the AL may use a lower value during lost block recovery. In the case when the GBT APDUs carry an unconfirmed service (BTS = FALSE, BTW = 0; see above), the value of the window shall be 0 indicating that no GBT APDUs received shall be confirmed (and hence no lost blocks can be recovered);
- the block-number (BN) field indicates the number of the block sent. The first block sent shall have block-number = 1. Block-number shall be incremented with each GBT APDU sent, even if block-data (BD) is empty. However, during lost block recovery a block number may be repeated;
- the block-number-acknowledged (BNA) field indicates the number of the block acknowledged. If no blocks have been lost, it shall be equal to the number of the last block received. However, if one or more blocks are lost, it shall be equal to the number of the block up to which no blocks are missing;
- the block-data (BD) field carries one part of the xDLMS APDU that is sent using the GBT mechanism.

#### 9.4.6.13.2 The GBT procedure

##### 9.4.6.13.2.1 Overview

The GBT procedure is shown in Figure 140. It can be:

- confirmed, to carry APDUs corresponding to the service primitives of any confirmed xDLMS service;
- unconfirmed, to carry an APDU corresponding to an unconfirmed service primitive from the client to the server or to carry an APDU corresponding to an unsolicited service request from the server to the client.

NOTE GBT is often used with the DataNotification and Access services.

The GBT procedure is essentially the same on the client and on the server side. It is called by the AL when:

- a) a service primitive – .request or .response – is invoked by the AP and either:
  - Invocation\_Type = COMPLETE or FIRST-PART with the GBT parameters BTS and BTW present; or
  - the encoded and cryptographically protected APDU to be sent is too long to fit into the maximum APDU size negotiated;
- b) a GBT APDU is received from the peer.

The GBT procedure comprises three sub-procedures:

- 1) *Send GBT APDU stream*, see 9.4.6.13.4;
- 2) *Process GBT APDU*, see 9.4.6.13.5;
- 3) *Check RQ and fill gaps*, see 9.4.6.13.6.

These sub-procedures are called by the AL as described in 9.4.6.13.2.2 and 9.4.6.13.2.3.

The model of the GBT procedure uses a Send Queue SQ and a Receive Queue RQ. Each block B in the queues has a block number BN, a flag LastBlock LB and a payload BlockData BD (that may be empty). The queues are ordered by BN.

The SQ is filled by the AL after processing the service primitives invoked by the AP i.e.:

- building the APDU;
- applying cryptographic protection as stipulated by the protection parameters;
- splitting the protected APDU into blocks.

For all service primitives, partial service invocations may be used as specified in 9.3.5, Figure 113 and 9.4.6.13, Figure 139. However, this has no consequence for the GBT protocol. The service Invocation\_Type may be COMPLETE, FIRST-PART, ONE-PART or LAST-PART. Each service invocation adds one or more blocks to the SQ. The block number BN is incremented every time a new block is added.

The blocks in the SQ are sent by the *Send GBT APDU stream* sub-procedure.

The RQ is filled by the *Process GBT APDU* sub-procedure. Gaps in the RQ – in the case of a confirmed GBT procedure – are filled by the *Check RQ and fill gaps* sub-procedure.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	349/614
-----------------------	------------	-----------------------	---------

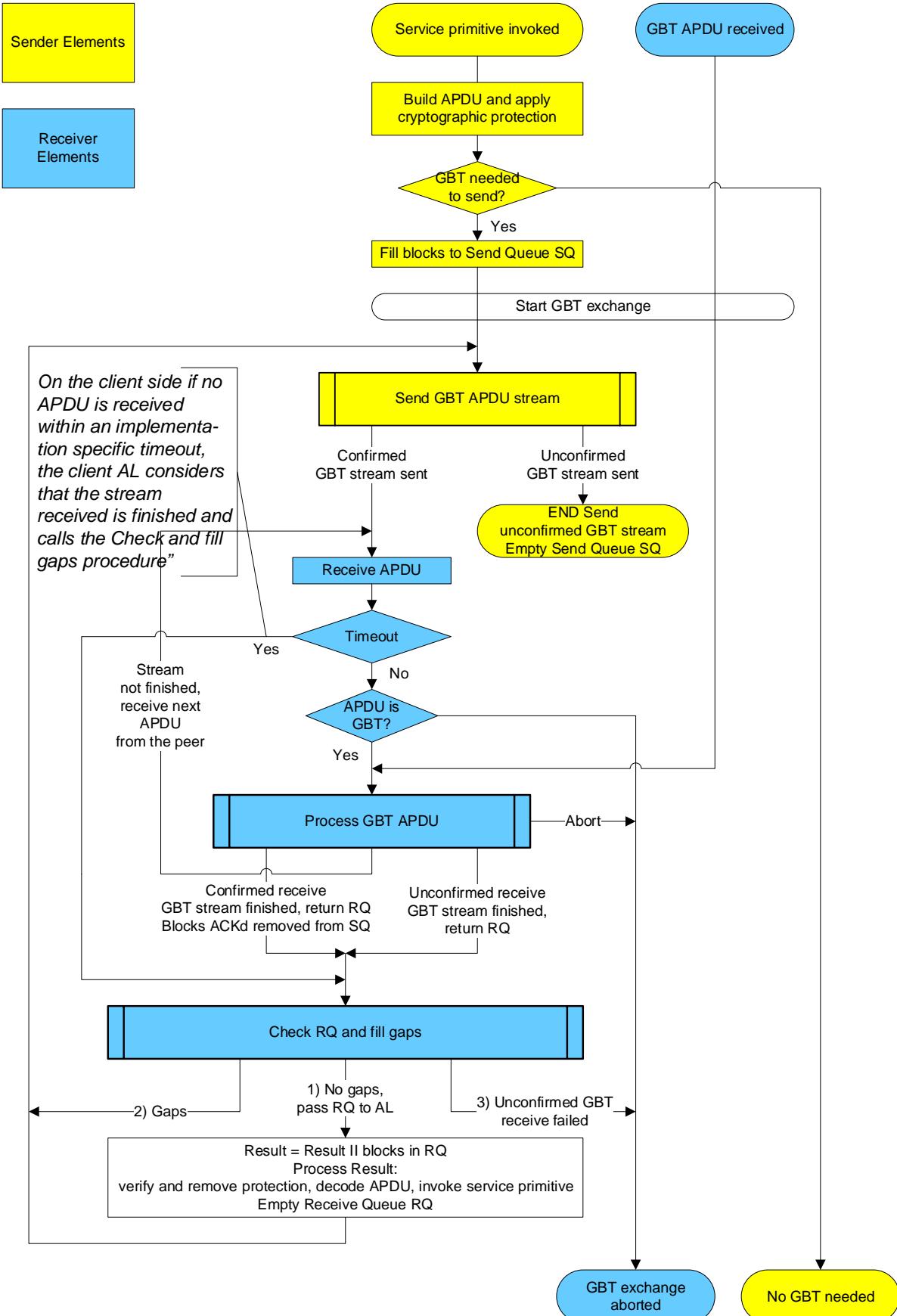


Figure 140 – The GBT procedure

#### 9.4.6.13.2.2 The confirmed GBT procedure

The confirmed GBT procedure may be triggered by a local service invocation or by the reception of a GBT APDU from the peer.

##### Confirmed GBT procedure triggered by a service invocation

When the AP invokes a service primitive and GBT is needed – see 9.4.6.13.2.1 a) – the AL fills the blocks to the SQ and invokes the *Send GBT APDU stream* sub-procedure; see 9.4.6.13.4. When the last block of a GBT stream has been sent the control is given back to the AL.

The AL listens then for an APDU from the peer.

On the client side if no APDU is received within an implementation specific timeout, the client AL considers that the stream received is finished and calls the *Check RQ and fill gaps* sub-procedure.

When an APDU is received the AL checks if it is a GBT APDU.

NOTE 1 This is determined from the tag of the APDU received.

If yes, the AL calls the *Process GBT APDU* sub-procedure; see 9.4.6.13.5. Otherwise, the GBT exchange is aborted. See also 9.4.6.13.8.

Once the GBT APDU received is processed, the control is given back to the AL:

- if the stream is not yet finished, the control is given back to the AL that waits for the next APDU from the peer;
- if the stream is finished, the AL calls the *Check RQ and fill gaps* sub-procedure; see 9.4.6.13.6.

If gaps are found the control is given back to the AL that calls again the *Send GBT APDU stream* sub-procedure.

If no gaps are found the control is given back to the AL and the blocks in the RQ are passed for processing:

- the blocks in the RQ are concatenated with result of the previous streams received;
- cryptographic protection is verified and removed;
- the APDU is decoded;
- the appropriate service primitive is invoked.

NOTE 2 Processing the result may lead to service invocations of Invocation\_Type COMPLETE, FIRST-PART, ONE-PART or LAST-PART.

The *Send GBT APDU* and the *Process GBT APDU / Check RQ and fill gaps* sub-procedures are called alternately both by the sender and the recipient until the GBT exchange is complete.

When the GBT procedure is used to transfer confirmed client-server services the server acknowledges the last block of the client, but the client does not acknowledge the last block of the server. Consequently:

- on the client side the procedure ends when the client LB has been ACKd and the server LB has been passed to the AL (i.e., all blocks from the server have been received without gaps);
- on the server side the procedure ends when the client LB has been ACKd, the server LB has been sent, and the client has not initiated lost block recovery.

When the GBT procedure is used to transfer an unsolicited, confirmed service – initiated by the server – the client acknowledges the last block of the server, but the server does not acknowledge the last block of the client. Consequently:

- on the server side the procedure ends when the server LB has been sent and the client LB has been received;
- on the client side the procedure ends when the server LB has been passed to the AL (i.e., all blocks from the server have been received without gaps) and the client LB has been sent;
- if the server does not receive the response, it may retry sending the request.

Confirmed GBT procedure triggered by reception of a GBT APDU from the peer

If a GBT procedure has not yet been started and the AL receives a GBT APDU, it calls the *Process GBT APDU sub-procedure*; see 9.4.6.13.5. From this point, the GBT procedure is started and continues as described above.

**9.4.6.13.2.3 The unconfirmed GBT procedure**

In this case, the sender sends a single stream using the *Send GBT APDU stream* sub-procedure and the recipient receives a single stream using the *Process GBT APDU* sub-procedure.

In the case when the GBT procedure is used to transfer an unconfirmed client-server service request and the stream cannot be completed, the client should resolve the situation.

In the case when the GBT procedure is used to transfer an unsolicited, unconfirmed request and the stream cannot be completed, the client – after a timeout – considers that the stream received from the server is finished and calls the *Check RQ and fill gaps* sub-procedure.

When the complete stream is received the AL calls the *Check RQ and fill gaps sub-procedure*; see 9.4.6.13.6.

If any blocks are missing, the blocks received are not processed and the unconfirmed GBT procedure ends.

An unconfirmed GBT procedure ends:

- on the sender side, when the last block has been sent;
- on the recipient side, when the last block has been received;
- when one or more gaps are found in the RQ.

**9.4.6.13.3 GBT procedure state variables**

The AL and the GBT procedure use several state variables, specified in Table 96.

**Table 96 – GBT procedure state variables**

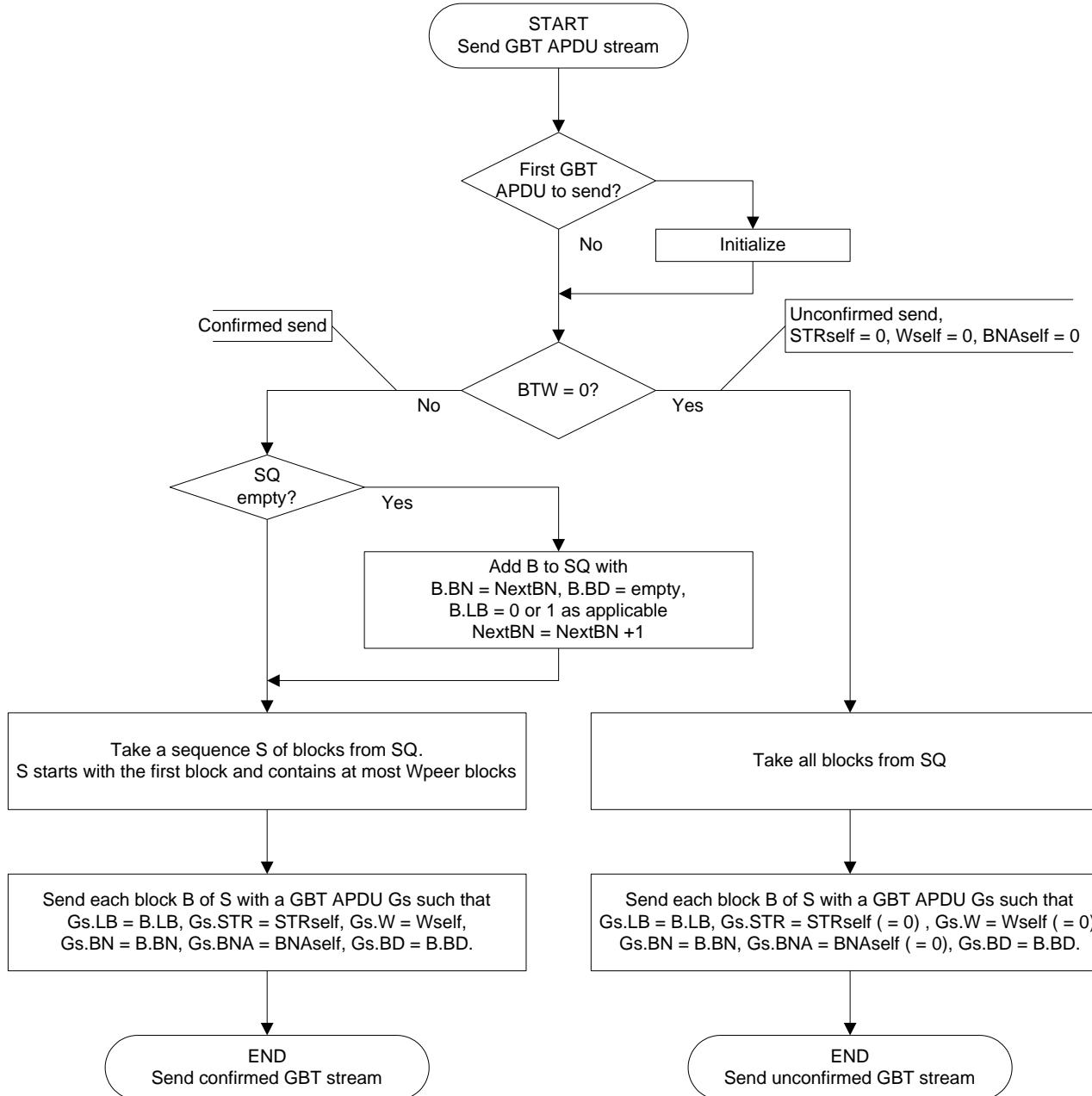
Name	Use	Initial value
BNAppear	<i>Send GBT stream</i> sub-procedure: –	–
	<i>Process GBT APDU</i> sub-procedure: The number of the block acknowledged by the peer. The value is taken from Gr.BNA.	–
BNAself	<i>Send GBT stream</i> sub-procedure: The number of the block acknowledged. The value is determined by the <i>Check RQ and fill gaps</i> sub-procedure and it is put in Gs.BNA. For an unconfirmed GBT stream the value is 0.	0
	<i>Process GBT APDU</i> sub-procedure: –	–
	<i>Check RQ and fill gaps</i> sub-procedure: Set to the B.BN of the block before the first gap found.	–
NextBN	The block number of the next block in the SQ. It is incremented every time a new block is added. Note that a new block may be added by the AL or – when the SQ is empty – by the <i>Send GBT stream</i> sub-procedure.	1
STRpeer	<i>Send GBT stream</i> sub-procedure: –	–
	<i>Process GBT APDU</i> sub-procedure: Indicates if the peer is streaming or not. The value is taken from Gr.STR.	–
STRself	<i>Send GBT stream</i> sub-procedure: The value to be put in Gs.STR. For an unconfirmed stream the value is FALSE.	BTS parameter of the service primitive invocation. (COMPLETE or FIRST-PART) (Default FALSE).

Name	Use	Initial value
	<i>Process GBT APDU</i> sub-procedure: –	–
Wpeer	<i>Send GBT stream</i> sub-procedure: The size of the streaming window of the peer, i.e. the maximum number of GBT APDUs that can be sent in a GBT stream. It is taken by the <i>Process GBT APDU</i> sub-procedure from Gr.W.	A value known a priori, or if not known, 1. It determines the size of the first GBT stream to be sent.
	<i>Process GBT APDU</i> sub-procedure: The size of the streaming window of the peer. It is taken from Gr.W and it is used by the <i>Send GBT stream</i> sub-procedure for the next GBT stream to be sent.	–
Wself	<i>Send GBT stream</i> sub-procedure: The size of the streaming window i.e. the maximum number of GBT APDUs the sender can receive from the peer in one stream. It is put in Gs.W. For an unconfirmed stream the value is 0.	BTW parameter of the service primitive invocation. (COMPLETE or FIRST-PART) Default = 1.
	<i>Process GBT APDU</i> sub-procedure: The number of GBT APDUs the recipient can receive in one GBT stream.	For a confirmed procedure, BTW parameter of the recipient (a value shared by the AP and the AL). For an unconfirmed procedure it is an implementation specific value.
	<i>Check RQ and fill gaps</i> sub-procedure: This is set to the number of missing blocks to be recovered. This can be any value from 1 up to the number of blocks to be recovered.	–

#### 9.4.6.13.4 Send GBT APDU stream sub-procedure

##### 9.4.6.13.4.1 General

The *Send GBT stream* sub-procedure – shown in Figure 141 – is used both for confirmed and unconfirmed GBT procedures.



Note to figure: See also 9.4.6.13.4.3.2 to determine the management of B.LB.

**Figure 141 – Send GBT APDU stream sub-procedure**

It sends a sequence S of blocks B to the peer. It is invoked by the AL when there are blocks in the SQ, or by the GBT procedure itself to acknowledge the blocks received in a stream and to send further blocks from the SQ.

#### 9.4.6.13.4.2 Initialization

Before the first GBT APDU is sent, the initial values shall be set according to Table 96.

During a confirmed GBT exchange the value of the BNAspeer state variable is updated by the *Process GBT APDU* sub-procedure. The value of the Wself and BNAself state variables are updated by the *Check RQ and fill gaps* sub-procedure.

For an unconfirmed GBT procedure the number of blocks the receiver is able to receive should be known a priori. The value of the STRself, Wself and BNAself state variables shall be set to 0.

#### 9.4.6.13.4.3 Confirmed GBT stream send

##### 9.4.6.13.4.3.1 General

For a confirmed GBT exchange, the AP invokes the service primitive with BTW > 0.

First, it is checked if the SQ is empty. On the client side this occurs when all blocks have been sent and acknowledged. In the server side this occurs if blocks are not yet available or all blocks have been sent and acknowledged. If the SQ is empty, an empty block is added to the SQ and NextBN is incremented.

##### 9.4.6.13.4.3.2 Last block management

The Last Block bit LB is controlled as follows.

When the GBT procedure is used with confirmed client-server services then:

- the client shall send each GBT APDU with LB = 0 unless it has no further blocks to send. When the last block is reached LB shall be set to 1. Once all blocks including the last block have been acknowledged by the server then any following client GBT APDUs – required for the purpose of acknowledging server GBT APDUs – shall be sent with an empty block and LB = 1;
- the server shall send each GBT APDU with LB = 0 until the last block of the client is received and the block to send is the last block of the server.

When the GBT procedure is used with an unsolicited, confirmed service – initiated by the server – then:

- the server shall send each GBT APDU with LB = 0 until it has no further blocks to send. When the last block is reached LB shall be set to 1;
- the client shall send each GBT APDU with LB = 0 until the last block of the server is received and the block to send is the last block of the client.

In any case, during lost block recovery, the LB property of each block is preserved: according to the model, a block is removed from the SQ when it is acknowledged by the peer. Therefore, if a block other than the last block is to be recovered, it shall be sent with LB = 0. If the last block is to be recovered, it shall be sent with LB = 1.

##### 9.4.6.13.4.3.3 Process

The sub-procedure takes a sequence S of blocks from the SQ. The sequence S starts with the first block and contains at most Wpeer blocks. Each block B of S is sent with a GBT APDU Gs such that Gs.LB = B.LB, Gs.STR = STRself, Gs.W = Wself, Gs.BN = B.BN, Gs.BNA = BNAsself and Gs.BD = B.BD.

The sub-procedure is finished once all blocks in S have been sent. The last GBT APDU Gs of the stream shall be sent with Gs.STR = FALSE. Gs.STR shall also be set to FALSE if B.LB = TRUE.

The blocks acknowledged by the peer are removed from the SQ by the *Process GBT APDU* sub-procedure. Therefore, when all blocks have been sent and acknowledged, the SQ is empty except that the blocks received with the last stream – that do not need to be acknowledged by the peer – remain in SQ and have to be removed when the end of the GBT exchange is detected.

#### 9.4.6.13.4.4 Unconfirmed GBT send

For an unconfirmed GBT send, the AP invokes the service primitive with BTW = 0 signifying that no blocks from the peer are accepted.

The sub-procedure sends the blocks in the SQ in a single stream. The SQ is emptied when the last block has been sent.

#### 9.4.6.13.5 Process GBT APDU sub-procedure

##### 9.4.6.13.5.1 General

The *Process GBT APDU* sub-procedure is shown in Figure 142.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	355/614
-----------------------	------------	-----------------------	---------

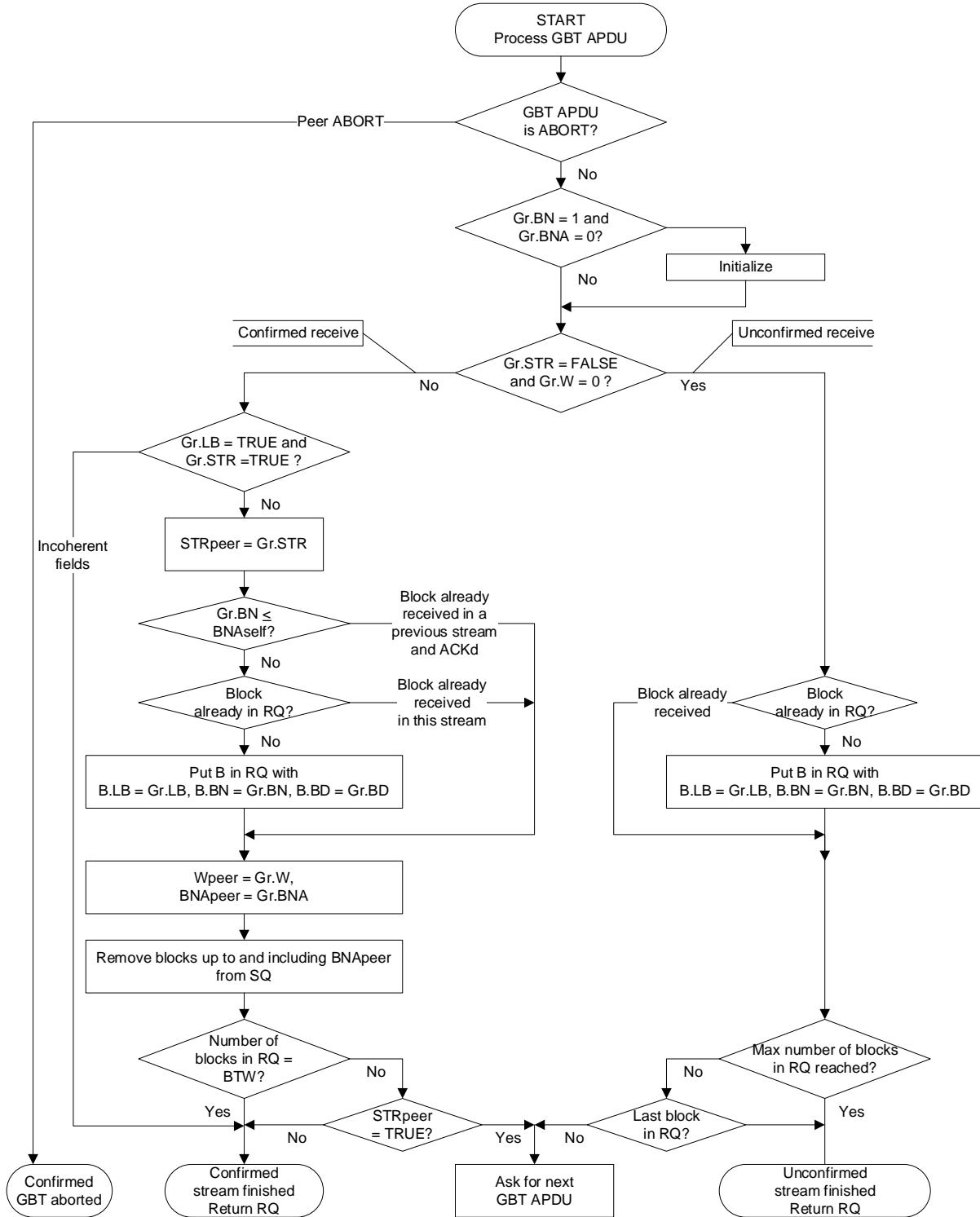


Figure 142 – Process GBT APDU sub-procedure

First, the value of Gr.STR and Gr.BN is checked. If the GBT APDU is an ABORT GBT APDU – see 9.4.6.13.8 – this indicates that the peer wants to abort the exchange.

Then, the value of Gr.BN and Gr.BNA is checked. If Gr.BN = 1 and Gr.BNA = 0, this indicates the start of a new GBT exchange and the initial values shall be set according to Table 96.

Then, the value of Gr.STR and Gr.W is checked. If Gr.STR = FALSE and Gr.W = 0, this indicates that the GBT procedure is unconfirmed; otherwise it is confirmed.

#### 9.4.6.13.5.2 Processing GBT APDUs in a confirmed GBT procedure

First, the values of Gr.LB and Gr.STR are checked. If Gr.LB = TRUE and Gr.STR = TRUE the values are incoherent: streaming should be finished when the last block is sent.

Then STRpeer is set to Gr.STR.

Then a check is made to determine if the block has already been received in a previous stream (Gr.BN  $\leq$  BNAsself) or in the current stream (block is already in RQ; this is determined if a value of B.BN in the RQ is equivalent to Gr.BN). In either cases, the duplicate block is not put to RQ.

Otherwise, the block B is put to the RQ with B.LB = Gr.LB, B.BN = Gr.BN B.BD = Gr.BD.

Wpeer is set to Gr.W and BNapeer is set to Gr.BNA.

The blocks up to and including BNapeer are removed from the SQ.

If the number of blocks in the RQ reaches Wself (BTW parameter of the recipient, a value shared by the AP and the AL), then the stream is considered to be finished.

Otherwise, the value of STRpeer is checked:

- if it is TRUE, then the control is given back to the AL that waits for the next APDU;
- if it is FALSE, the stream is considered to be finished.

When the stream is finished no more GBT APDUs are listened for. Any GBT APDUs received are considered as overflow and shall not be acknowledged.

The blocks in the RQ are passed to the AL that calls the *Check RQ and fill gaps* sub-procedure.

#### 9.4.6.13.5.3 Processing GBT APDUs in an unconfirmed GBT procedure

First, the value of Gr.BN is checked. If the block is already in the RQ it is discarded. Otherwise, it is put to the RQ with B.LB = Gr.LB, B.BN = Gr.BN B.BD = Gr.BD.

If the number of the blocks in the RQ reaches the maximum number of blocks that can be accommodated, then the stream is considered to be finished.

Otherwise, it is checked if the last block (with B.LB = TRUE) is in the RQ:

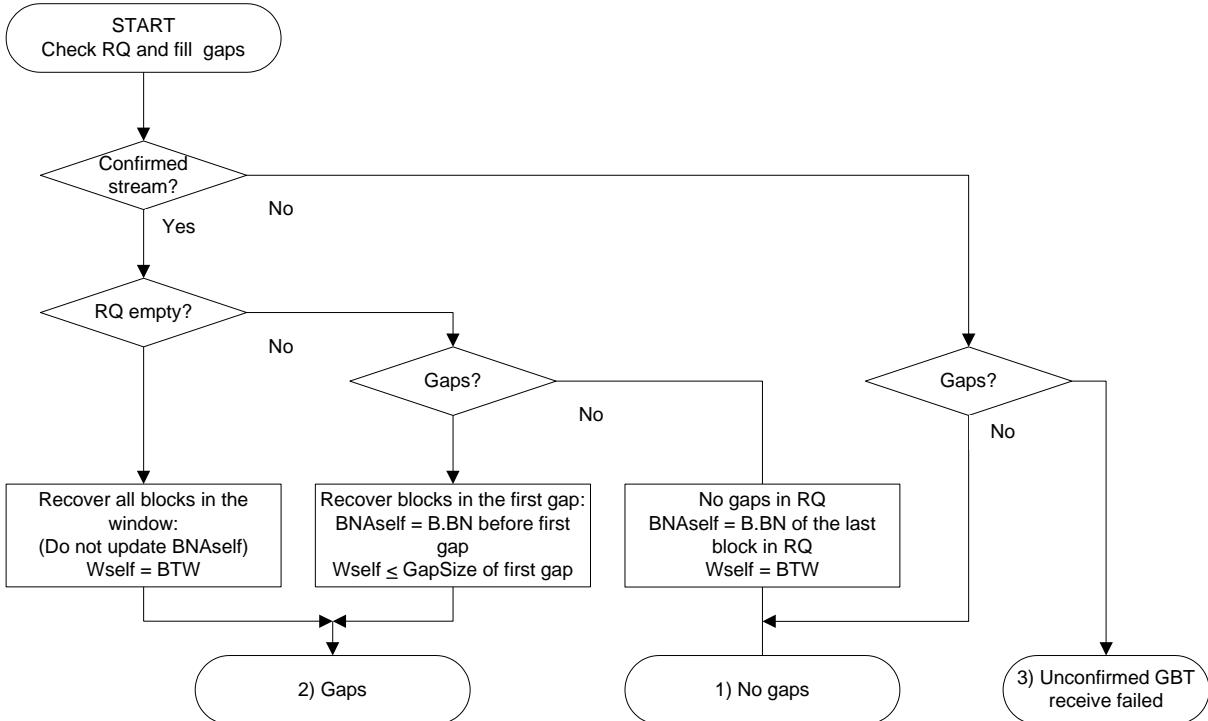
- if no, the control is given back to the AL that waits for the next APDU;
- if yes, the stream is considered to be finished.

When the stream is finished no more GBT APDUs are listened for. Any GBT APDUs received are considered as overflow and shall not be acknowledged. The blocks in the RQ are passed to the AL, the RQ is emptied and the unconfirmed receive GBT procedure is ended.

#### 9.4.6.13.6 Check RQ and fill gaps sub-procedure

##### 9.4.6.13.6.1 General

The *Check RQ and fill gaps* sub-procedure is shown in Figure 143.

**Figure 143 – Check RQ and fill gaps sub-procedure**

#### 9.4.6.13.6.2 Confirmed GBT procedure

This subclause describes one possible recovery strategy. Other strategies that result in the recovery of the missing blocks may also be used.

If the GBT stream is confirmed, first it is checked if the RQ is empty. This occurs if no GBT APDUs have been received or all GBT APDUs in the stream have been discarded. Notice that the stream received may contain only one GBT APDU. In this case, BNAself is not updated, Wself is set to BTW and the control is given back to the AL.

If the RQ is not empty, it is checked for gaps:

- if gaps are found then the GBT procedure attempts to recover the blocks in the first gap found: BNAself is set to B.BN before the first gap, Wself is set to the number of blocks to be recovered and the control is given back to the AL that calls the *Send GBT APDU stream* sub-procedure;
- if no gaps are found, then BNAself is set to B.BN of the last block in the RQ, Wself is set to BTW and the control is given back to the AL.

#### 9.4.6.13.6.3 Unconfirmed GBT procedure

If the GBT procedure is unconfirmed, then the RQ is checked for gaps. If no gaps are found, then the control is given back to the AL and the blocks in the RQ are passed to the AL. Otherwise all blocks are discarded.

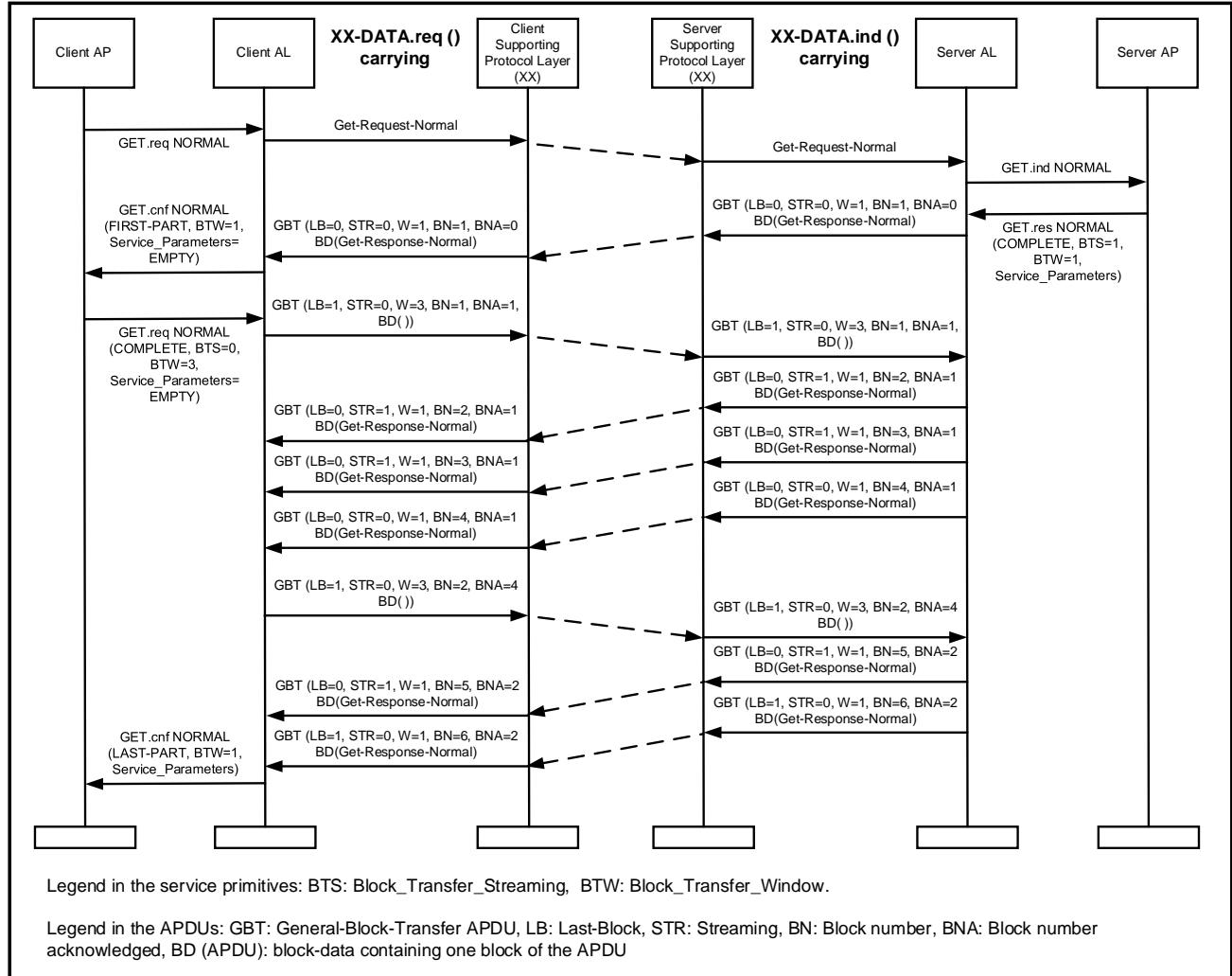
#### 9.4.6.13.7 GBT protocol examples

The protocol of the GBT mechanism is further explained with the help of Figure 144, Figure 145, Figure 146, Figure 147, Figure 148, Figure 149 and Figure 150. In these examples, it is assumed that both parties support GBT and six blocks are required to transfer the complete response or request (except in the DataNotification example, where four blocks are required).

NOTE 1 In these examples the service-specific block transfer mechanism is not used.

Abbreviations used on the Figures:

- BTS: Block\_Transfer\_Streaming, BTW: Block\_Transfer\_Window;
- GBT: general-block-transfer APDU;
- LB: last-block;
- STR: Streaming;
- BN: block-number, BNA: block-number-acknowledged;
- BD (APDU): block-data containing one block of the APDU.

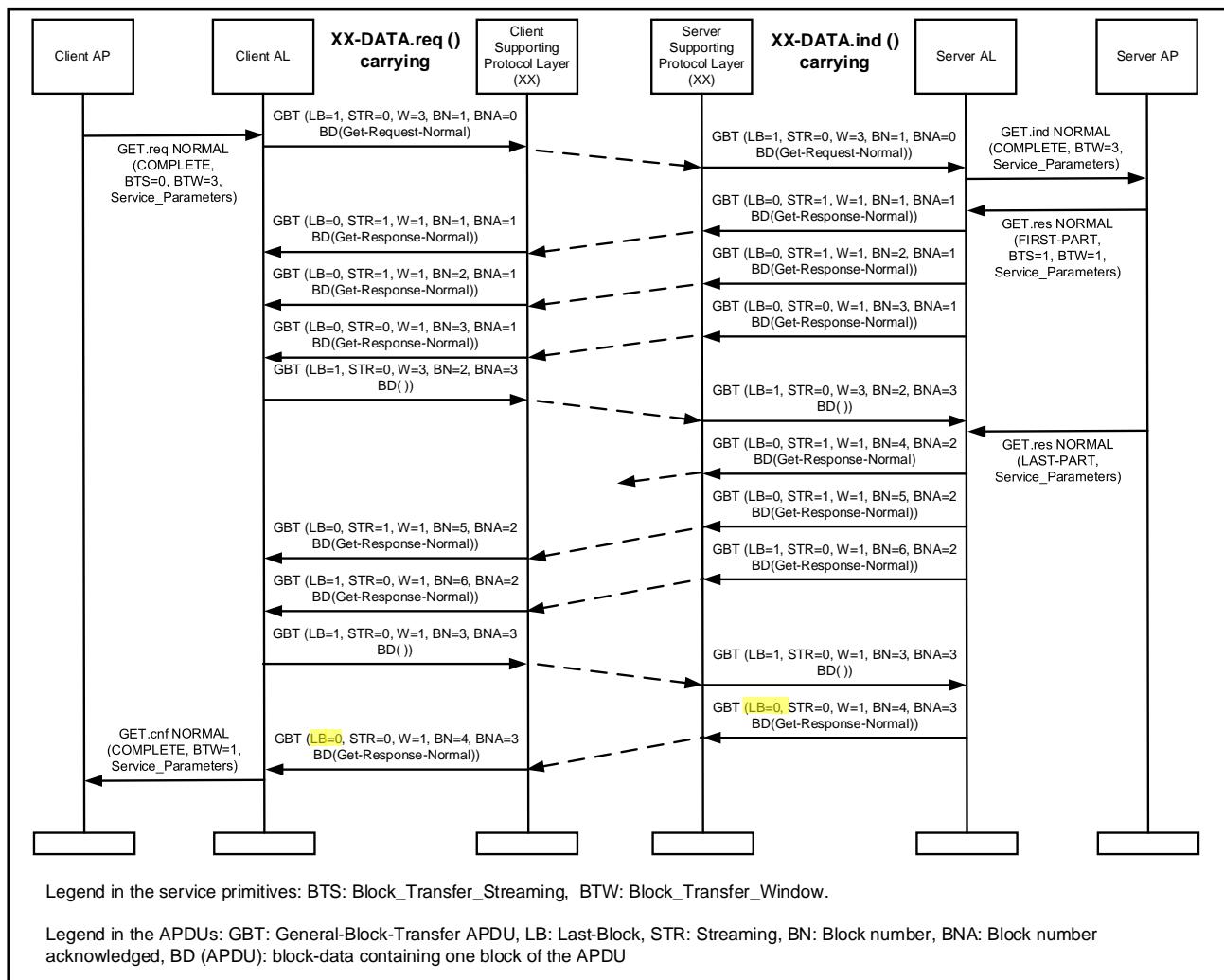


**Figure 144 – GET service with GBT, switching to streaming**

Figure 144 shows a GET service using GBT. After receiving the first GBT APDU, the client informs the server that it supports streaming. The server switches then to streaming. The process is the following:

- the client AP invokes a GET.request NORMAL service primitive, without additional service parameters. The client AL sends the request in a Get-Request-Normal APDU;
- the GET.response service parameters are long so the server AP invokes a GET.response NORMAL service primitive with additional service parameters: Invocation\_Type = COMPLETE, BTS = 1, BTW = 1 meaning that the server allows sending block streams, but it does not accept block streams from the client. The server AL sends a GBT APDU, containing the first block of the response;
- the client AL invokes a GET.confirm NORMAL service primitive, Invocation\_Type = FIRST-PART, BTW = 1. The Service\_Parameters are empty. With this, it informs the AP that the response from the server is long;
- the client AP invokes a GET.request NORMAL service primitive, with Invocation\_Type = COMPLETE, BTS = 0, BTW = 3, to advertise its capabilities to receive block streams. Note that

- the Service\_Parameters are empty, as these have been passed already in the first GET.request NORMAL service invocation. The client AL sends a GBT APDU. The last-block bit in the APDU is set to 1 and the streaming bit is set to 0 as the client has no blocks to send;
- the server sends then the 2<sup>nd</sup> (STR = 1, BN = 2), 3<sup>rd</sup> (STR = 1, BN = 3) and 4<sup>th</sup> (STR = 0, BN = 4) blocks;
  - the client AL sends a GBT APDU to confirm the reception of the 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> block (LB = 1, STR = 0, W = 3, BNA = 4);
  - the server AL sends the 5<sup>th</sup> (STR = 1, BN = 5) and the 6<sup>th</sup>, last block (LB = 1, STR = 0, BN = 6);
- NOTE 2 The last block **of the server** is not confirmed. However, when lost, it can be recovered. See Figure 147.
- the client AL invokes a GET.confirm NORMAL service primitive with Invocation\_Type = LAST-PART. The service parameters include the complete response to the GET.request.



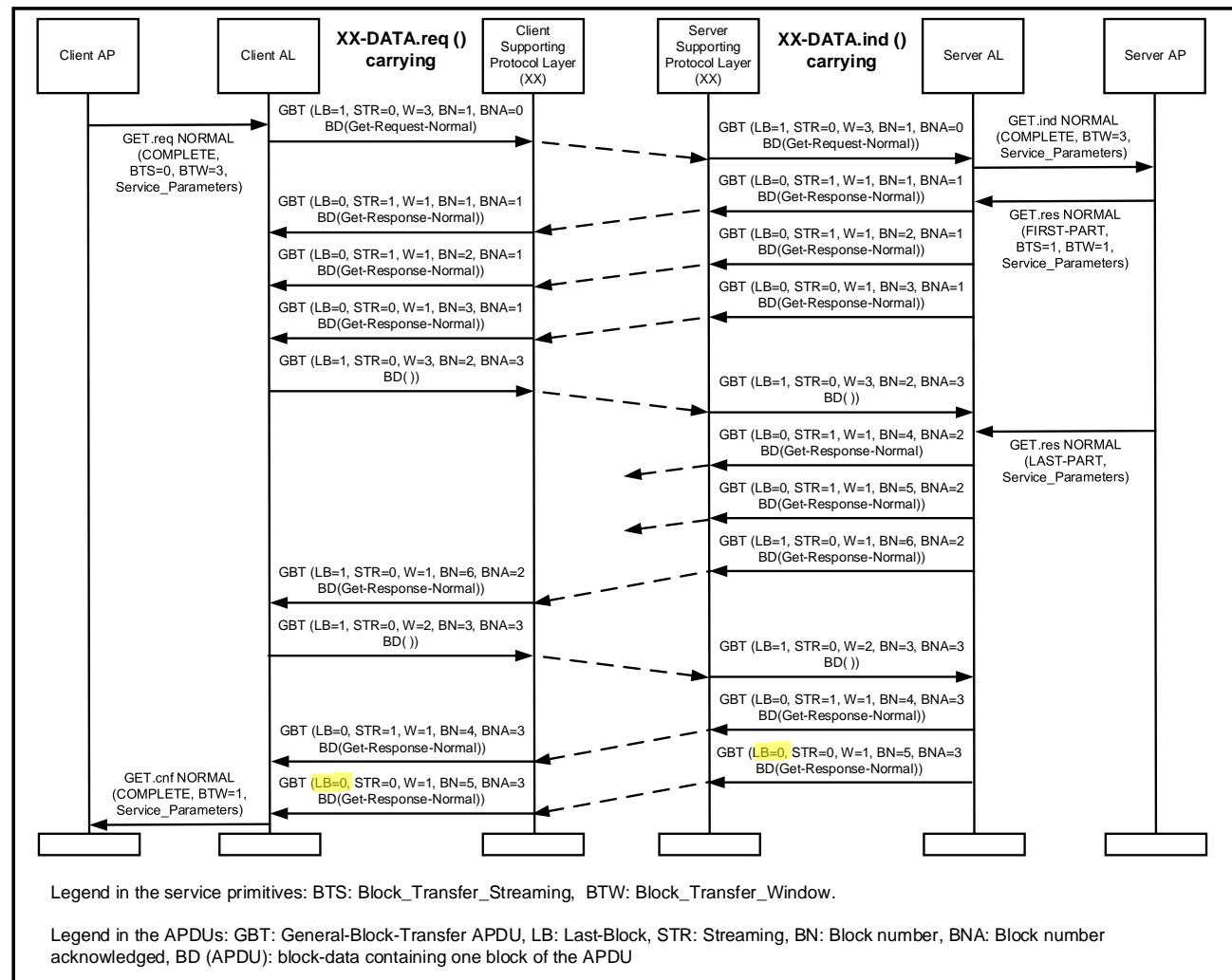
**Figure 145 – GET service with partial invocations, GBT and streaming, recovery of 4<sup>th</sup> block sent in the 2nd stream**

Figure 145 shows **transporting** a GET service using GBT, with partial service invocations on the server side and streaming. The client advertises in the first request its streaming capabilities (BTW = 3; BTW > 1 means that block streams can be received). The 4<sup>th</sup> block, sent in the second stream by the server is lost and it is recovered. The process is the following:

- the client AP invokes a GET.request NORMAL service primitive, with Invocation\_Type = COMPLETE, BTS = 0, BTW = 3. The client AL sends a GBT APDU with STR = 0, Window = 3. The

- server AL invokes the GET.indication NORMAL service primitive with Invocation\_Type = COMPLETE, BTW = 3;
- the server AP invokes a GET.response NORMAL service primitive with Invocation\_Type = FIRST-PART, BTS = 1, BTW = 1. Service\_Parameters include the first part of the response. The server AL sends the 1<sup>st</sup>, 2<sup>nd</sup> and 3<sup>rd</sup> block;
  - the client AL sends a GBT APDU to confirm the reception of the three blocks. The server AP invokes a GET.response NORMAL service primitive with Invocation\_Type = LAST-PART. Service\_Parameters include the second, last part of the response. The server AL sends the 4<sup>th</sup>, 5<sup>th</sup> and 6<sup>th</sup> block (LB = 1, STR = 0, BN = 6). However, the 4<sup>th</sup> block gets lost;
  - the client AL indicates that the 4<sup>th</sup> block has not been received by sending a GBT APDU confirming the reception of the 3<sup>rd</sup> block (STR = 0, window = 1, BNA = 3). Notice that the client AL drops down the window size to 1 to indicate that only one block has to be re-sent;
  - the server sends again the 4<sup>th</sup> block (LB = 0, STR = 0, BN = 4, BNA = 3);
  - as the client has already received now all blocks, it invokes a GET.confirm NORMAL service primitive with Invocation\_Type = COMPLETE, BTW = 1. The Service\_Parameters of this invocation contain the complete response to the GET.request.

Figure 146 shows a scenario which is essentially the same as in Figure 145 except that the 4<sup>th</sup> and 5<sup>th</sup> blocks are lost and recovered.



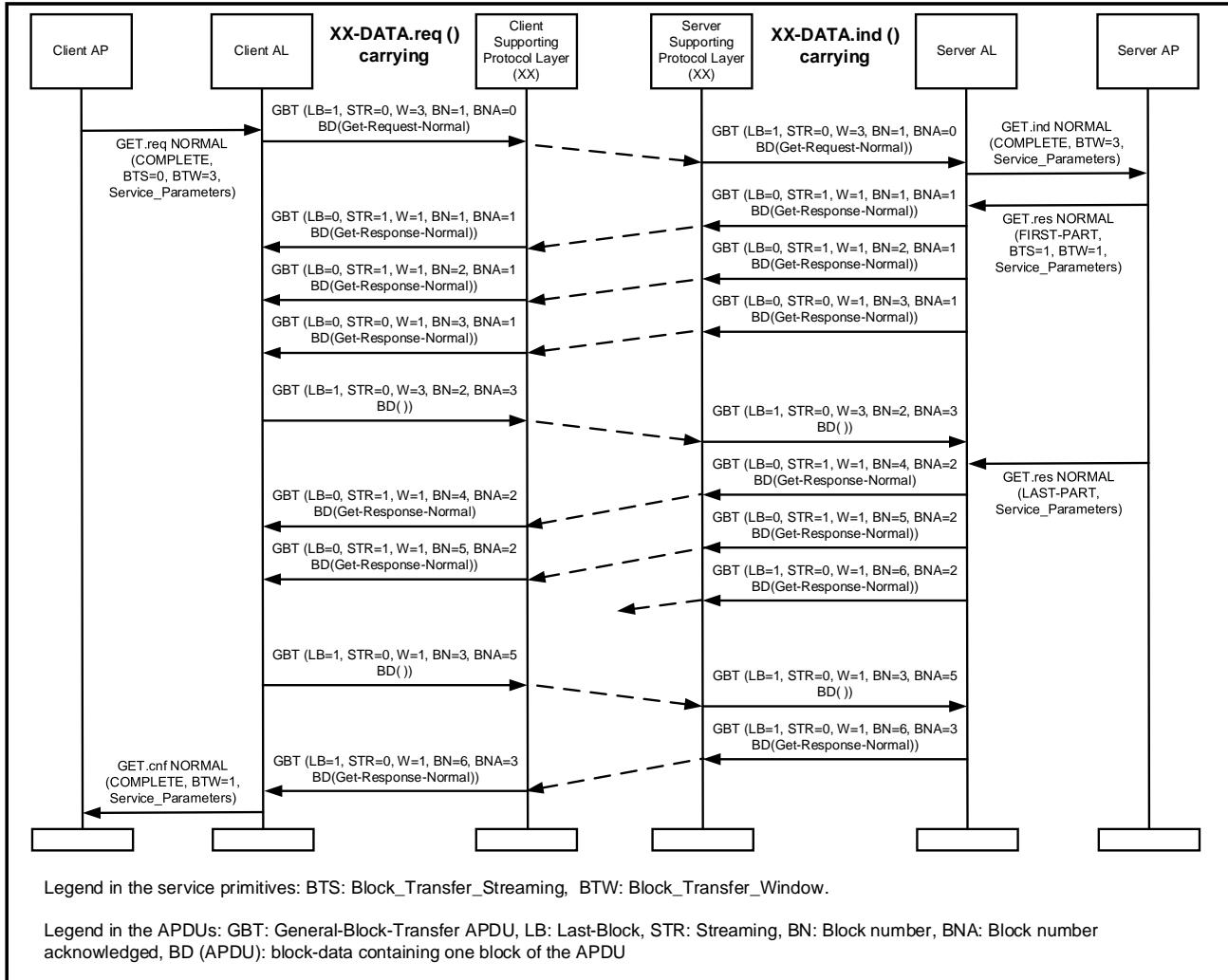
**Figure 146 – GET service with partial invocations, GBT and streaming, recovery of 4<sup>th</sup> and 5<sup>th</sup> block**

The process is the following:

- the client receives the 6<sup>th</sup> block (LB = 1, STR = 0, BN = 6, BNA = 2);

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	361/614
-----------------------	------------	-----------------------	---------

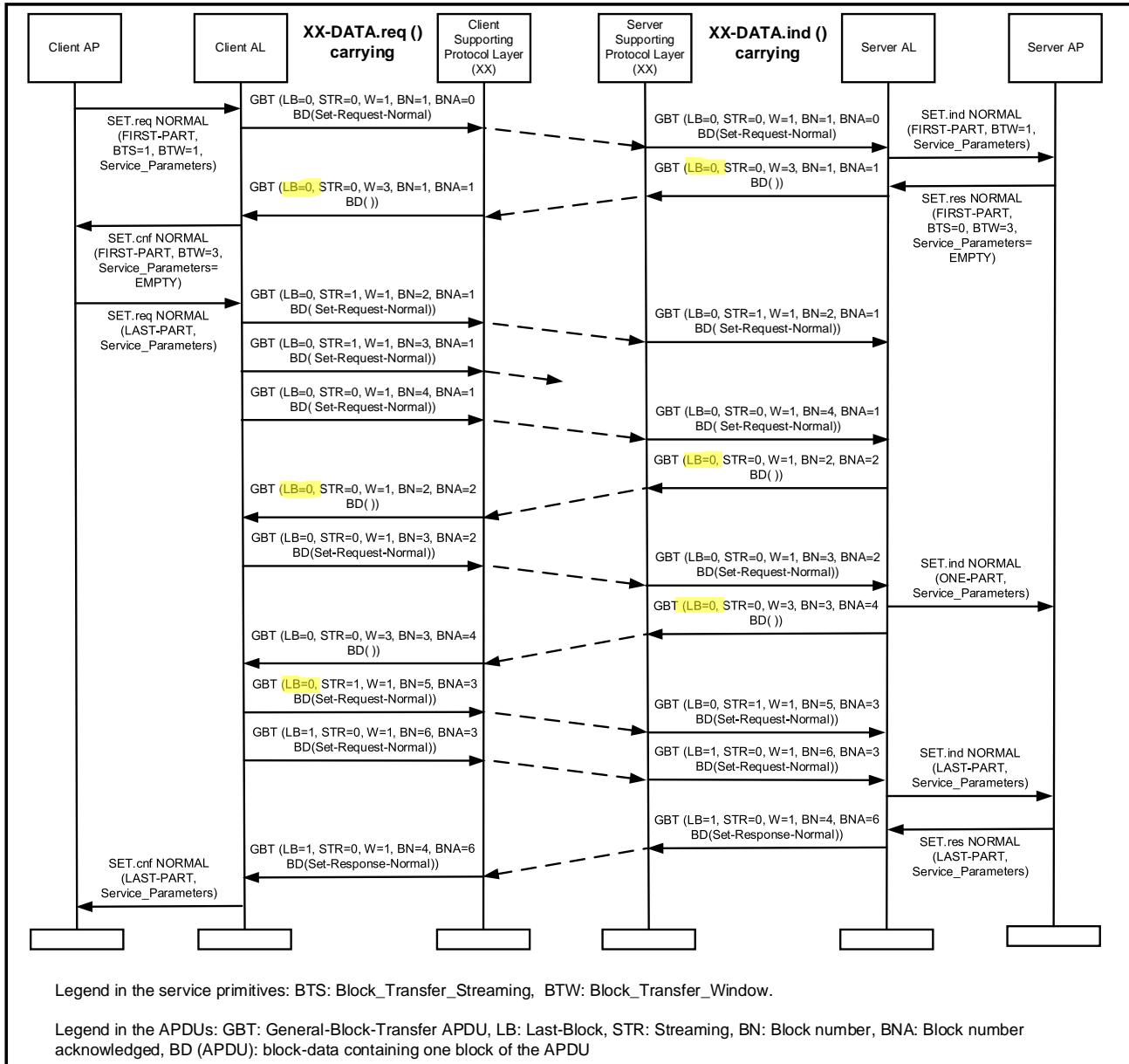
- the client indicates that the 4<sup>th</sup> and the 5<sup>th</sup> blocks have been lost, by sending a GBT APDU with W = 2, BNA = 3, i.e. meaning that no blocks are missing up to the 3<sup>rd</sup> block but two blocks have been lost and that the server can send these two using streaming;
- the server sends then the lost (not confirmed) 4<sup>th</sup> (LB = 0, STR = 1, BN = 4) and 5<sup>th</sup> block (LB = 0, STR = 0, BN = 5).
- the client has received now each block. It invokes then a GET.confirm NORMAL service primitive with Invocation\_Type = COMPLETE, BTW = 1. The Service\_Parameters include the parameters of the complete response to the GET.request.



**Figure 147 – GET service with partial invocations, GBT and streaming, recovery of last block**

Figure 147 shows a scenario when the last block sent in the second stream gets lost and is recovered. The process is the following:

- the client receives the 5<sup>th</sup> block carried by a GBT APDU (LB = 0, STR = 1, BN = 5);
- as this is not the last block, after an implementation specific timeout the client sends a GBT APDU (LB = 1, STR = 0, BN = 3, BNA = 5);
- the server sends then the lost (not confirmed) 6<sup>th</sup> block carried by a GBT APDU (LB = 1, STR = 0, W = 1, BN = 6 and BNA = 3);
- when the client receives this APDU, it invokes a GET.confirm NORMAL service primitive with Invocation\_Type = COMPLETE, BTW = 1. The Service\_Parameters include the parameters of the complete response to the GET.request.



**Figure 148 – SET service with GBT, with server not supporting streaming, recovery of 3rd block**

Figure 148 shows transporting a SET service with GBT and streaming. The 3<sup>rd</sup> block sent by the client is lost and recovered. The process is the following:

- the client AP invokes a SET.request NORMAL service primitive with Invocation\_Type = FIRST-PART, BTS = 1, BTW = 1. The Service\_Parameters include the first part of the SET.request. The client AL sends the first block only because it does not know the streaming window size supported by the server;
- the server AL invokes a SET.indication NORMAL service primitive with Invocation\_Type = FIRST-PART, BTW = 1. The Service\_Parameters include the first part of the parameters of the SET.request;
- the server AP responds with a SET.response NORMAL service primitive with Invocation\_Type = FIRST-PART, BTS = 0, BTW = 3. The Service\_Parameters are empty. The server AL sends a GBT APDU with LB = 0, STR = 0, Window = 3; block-data is empty. From this, the client knows that the server can receive block streams and the window size = 3. Therefore it sends the 2<sup>nd</sup>, 3<sup>rd</sup> and 4<sup>th</sup> block in a stream. However, the 3<sup>rd</sup> block gets lost;

- the server indicates that block 3 is lost by confirming the reception of the 2<sup>nd</sup> block and dropping down the window size to 1 (LB = 0, STR = 0, W = 1, BN = 2, BNA = 2). The client sends then again the 3rd block (LB = 0, STR = 0, BN = 3, BNA = 2);
- the server AL invokes a SET.indication NORMAL service primitive with Invocation\_Type = ONE-PART. The server AL confirms the reception of the blocks up to the 4<sup>th</sup> block (LB = 0, STR = 0, W = 3, BNA = 4). Notice that the window size has been raised again to 3;
- the client sends then the 5<sup>th</sup> and the 6<sup>th</sup> block using streaming;
- when the server AL receives the 6<sup>th</sup>, last block it invokes a SET.indication NORMAL service primitive with Invocation\_Type = LAST-PART. Service\_Parameters include the last part of the parameters of the SET.request;
- the server AP invokes a SET.response NORMAL service primitive with Invocation\_Type = LAST-PART, and with the Service\_Parameters containing the result of the set operation(s). This is sent by the server AL in a GBT APDU with LB = 1. The client AL invokes the SET.confirm NORMAL service primitive with Invocation\_Type = LAST-PART. Service\_Parameters include the result of the set operations.

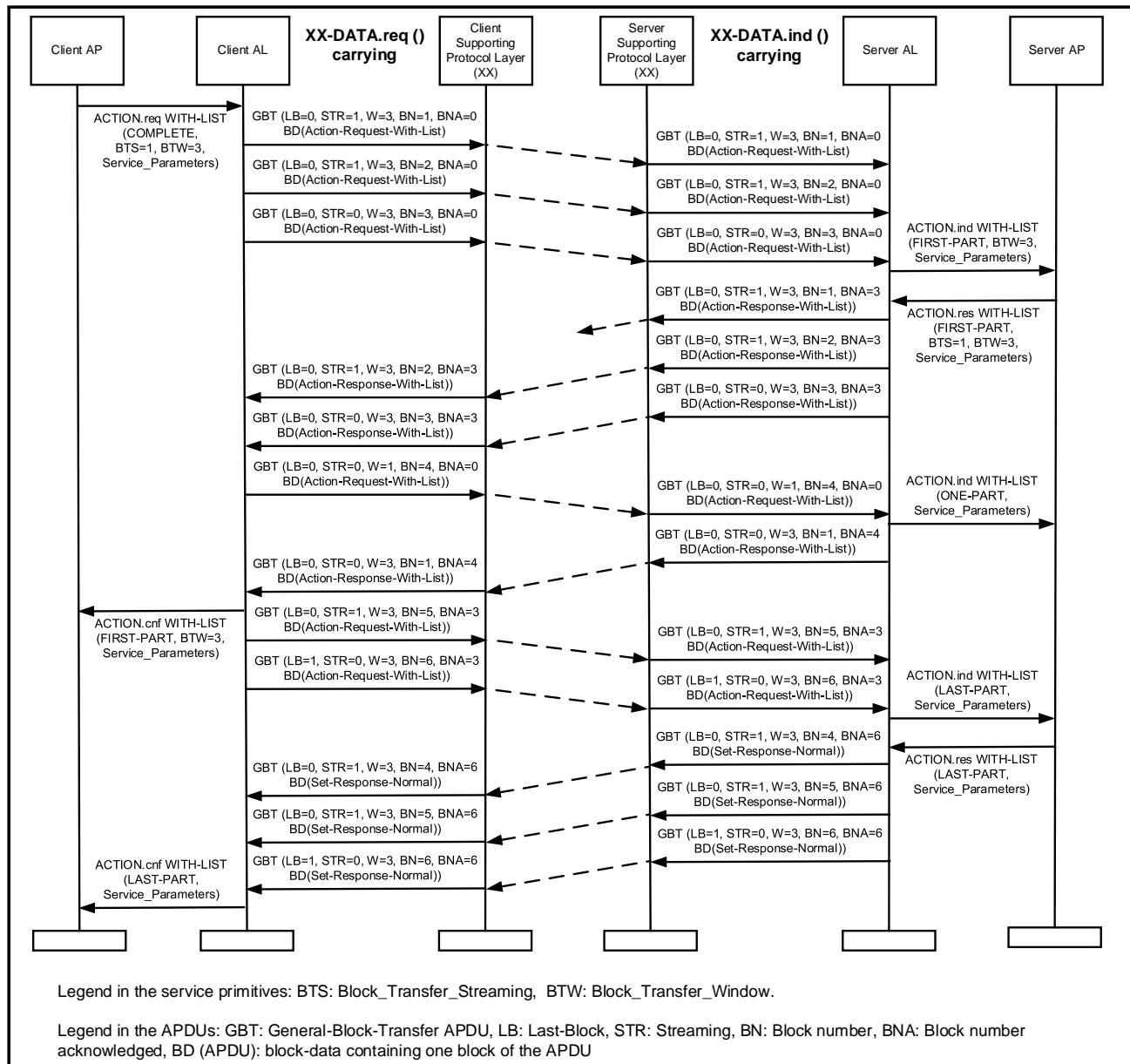
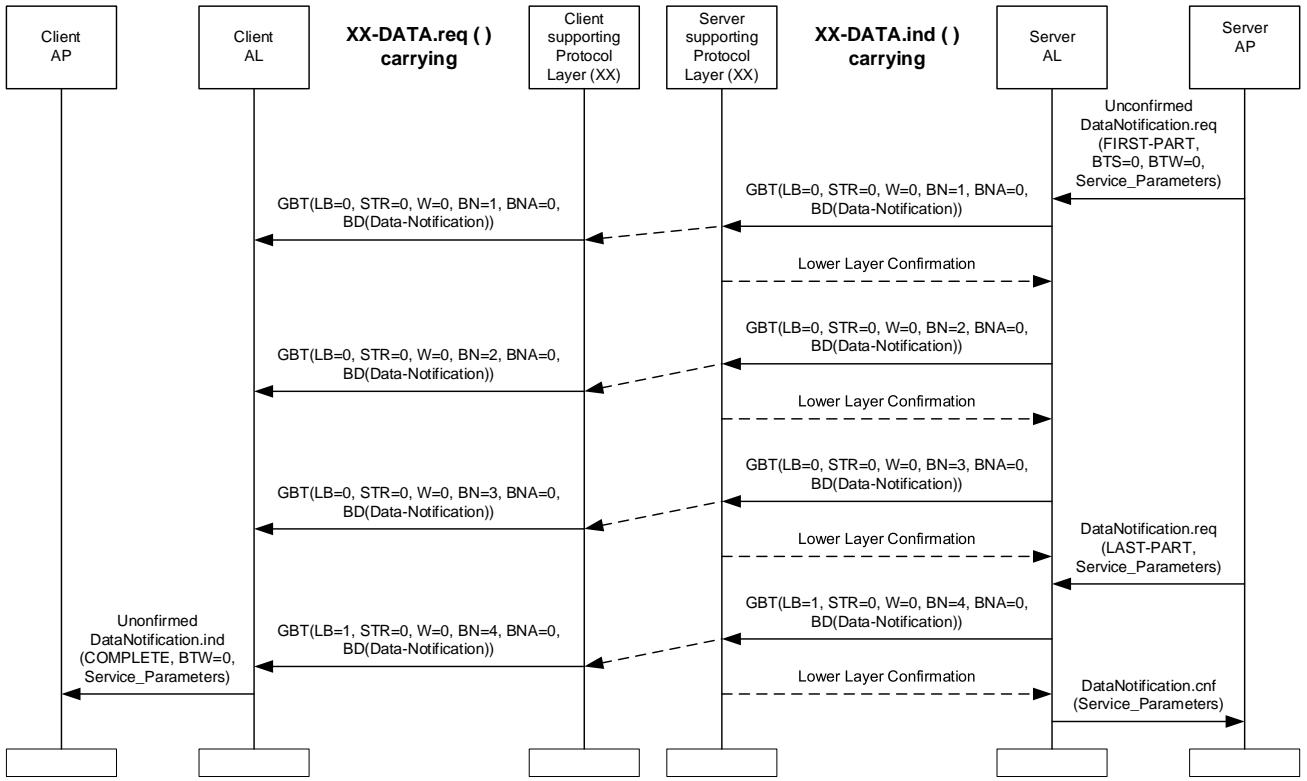


Figure 149 – ACTION-WITH-LIST service with bi-directional GBT and block recovery

Figure 149 shows **transporting** an ACTION-WITH-LIST service with partial service invocations, bidirectional block transfer and streaming. Both parties know *a priori* that the other party supports streaming with window size = 3. The first block sent by the server is lost and recovered. The process is the following:

- the client invokes an ACTION.request of type WITH-LIST service primitive with Invocation\_Type = COMPLETE, BTS = 1, BTW = 3. The client AL sends the first three blocks that carry a part of this request to the server. The server AL invokes an ACTION.indication of type WITH-LIST service primitive with Invocation\_Type = FIRST-PART, BTW = 3. Service parameters contain the first part of the request;
- the server AP processes this request and has the first part of the response available. It invokes an ACTION.response of type WITH-LIST service primitive with Invocation\_Type = FIRST-PART, BTS = 1, BTW = 3. The server AL sends this in three blocks using streaming. However, the 1<sup>st</sup> block is lost;
- the client AL asks the server to send the lost 1<sup>st</sup> block again by not confirming any blocks received. It also sends its 4<sup>th</sup> block (LB = 0, STR = 0, W = 1, BN = 4, BNA = 0). Notice that the client AL has dropped down the window size to 1;
- the server AL invokes an ACTION.indication of type WITH-LIST service primitive with INVOCATION\_Type = ONE-PART. Service\_Parameters contain one part of the request;
- the server sends the lost 1<sup>st</sup> block and confirms the 4<sup>th</sup> block received from the client (BN = 1, BNA = 4);
- the client AL invokes an ACTION.confirm of type WITH-LIST service primitive with additional parameters: Invocation\_Type = FIRST-PART, BTW = 3. The Service\_Parameters include one part of the response from the server;
- the client AL sends the 5<sup>th</sup> and the 6<sup>th</sup>, last block. Window size is raised again to 3;
- the server AL invokes an ACTION.indication of type WITH-LIST service primitive with Invocation\_Type = LAST-PART and with Service\_Parameters containing the last part of the ACTION.request;
- the server AP processes this and invokes an ACTION.response of type WITH-LIST service primitive with Invocation\_Type = LAST-PART; the Service\_Parameters contain the remaining part of the response. This is sent to client in three blocks using streaming;
- the client AL invokes an ACTION.confirm of type WITH-LIST service primitive with Invocation\_Type = LAST-PART. The Service\_Parameters include the last part of the response from the server.



Legend in the service primitive: BTS: Block\_Transfer\_Streaming, BTW: Block\_Transfer\_Window. Legend in the APDUs: GBT: General-Block-Transfer APDU, LB: Last-Block, STR: Streaming, W: Window, BN: Block number, BNA: Block number acknowledged, BD (APDU): block-data containing one block of the APDU

**Figure 150 – Unconfirmed DataNotification service with GBT with partial invocation**

Figure 150 shows transporting an unconfirmed DataNotification service with GBT, with partial service invocations on the server side. On this Figure, lower layer confirmations are also shown. After sending a block, a lower layer confirmation may be needed before the next block can be sent.

**NOTE** This applies to or GBT related MSCs.

The process is the following:

- the server AP invokes an **unconfirmed** DataNotification.request service primitive with Invocation\_Type = FIRST-PART, BTS = 0, BTW = 0. The Service\_Parameters include one part of the DataNotification.request;
- the server AL sends the GBT APDUs to the client. The reception of the blocks is not confirmed, block recovery is not available;
- when the client AL receives the last block, it assembles the block-data together and invokes a DataNotification.indication service primitive with Invocation\_Type = COMPLETE, BTW = 0. The Service\_Parameters include the complete DataNotification service parameters.

Notice that throughout the process the GBT APDUs are sent with STR = 0 and W = 0, signifying that the process is unconfirmed.

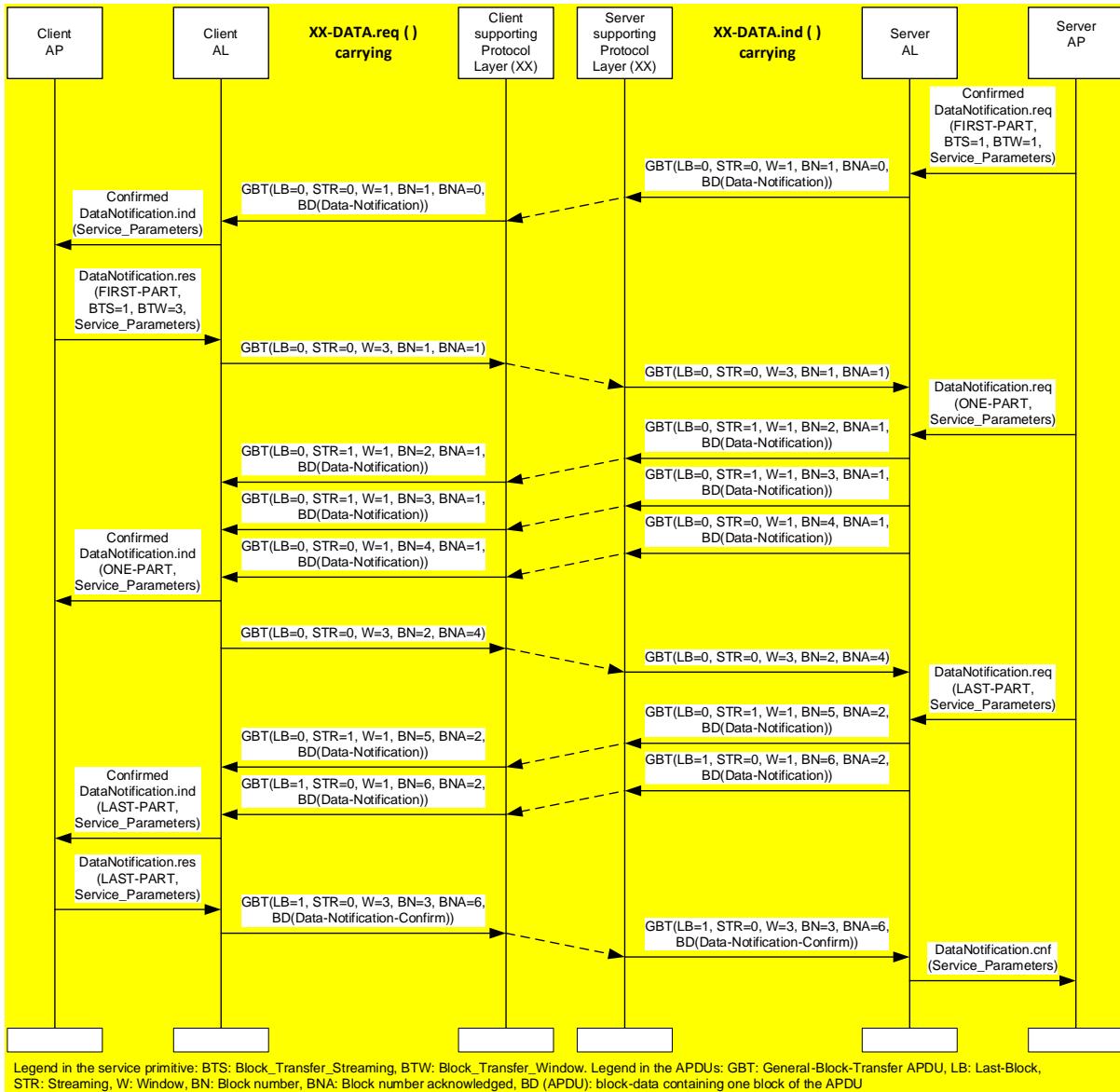
**Figure 151 – Confirmed DataNotification service with GBT**

Figure 151 shows transporting a Confirmed DataNotification service with GBT.

The process is the following:

- the server AP invokes the confirmed DataNotification.req service primitive with Invocation\_Type = FIRST\_PART, BTS =1, BTW = 1. The Service\_parameters include the first part of the DataNotification.request;
- the server AL sends one block to the client (as it does not yet know client window size);
- the client AL invokes a confirmed DataNotification.ind;
- the client AP responds with a DataNotification.res with Invocation\_Type = FIRST\_PART and BTW =3. The service parameters are empty;
- the client AL sends the GBT APDU to the server with W = 3, BN = 1 and BNA = 1;
- the server then sends the remaining blocks with further partial invocations and with streams of up to 3 blocks;
- the client AL acknowledges the blocks at the end of each stream. When the last block from the server is received, the client AL invokes the confirmed DataNotificaiton.ind with Invocation\_Type = LAST-PART;
- the Client AP invokes the confirmed DataNotification.res with Invocation\_Type = LAST-PART;
- the client AL sends the last block with LB = 1, confirming the last block of the server. BlockData contains the data-notification-confirm APDU;

- the server AL invokes DataNotification.cnf with Invocation\_type = COMPLETE.

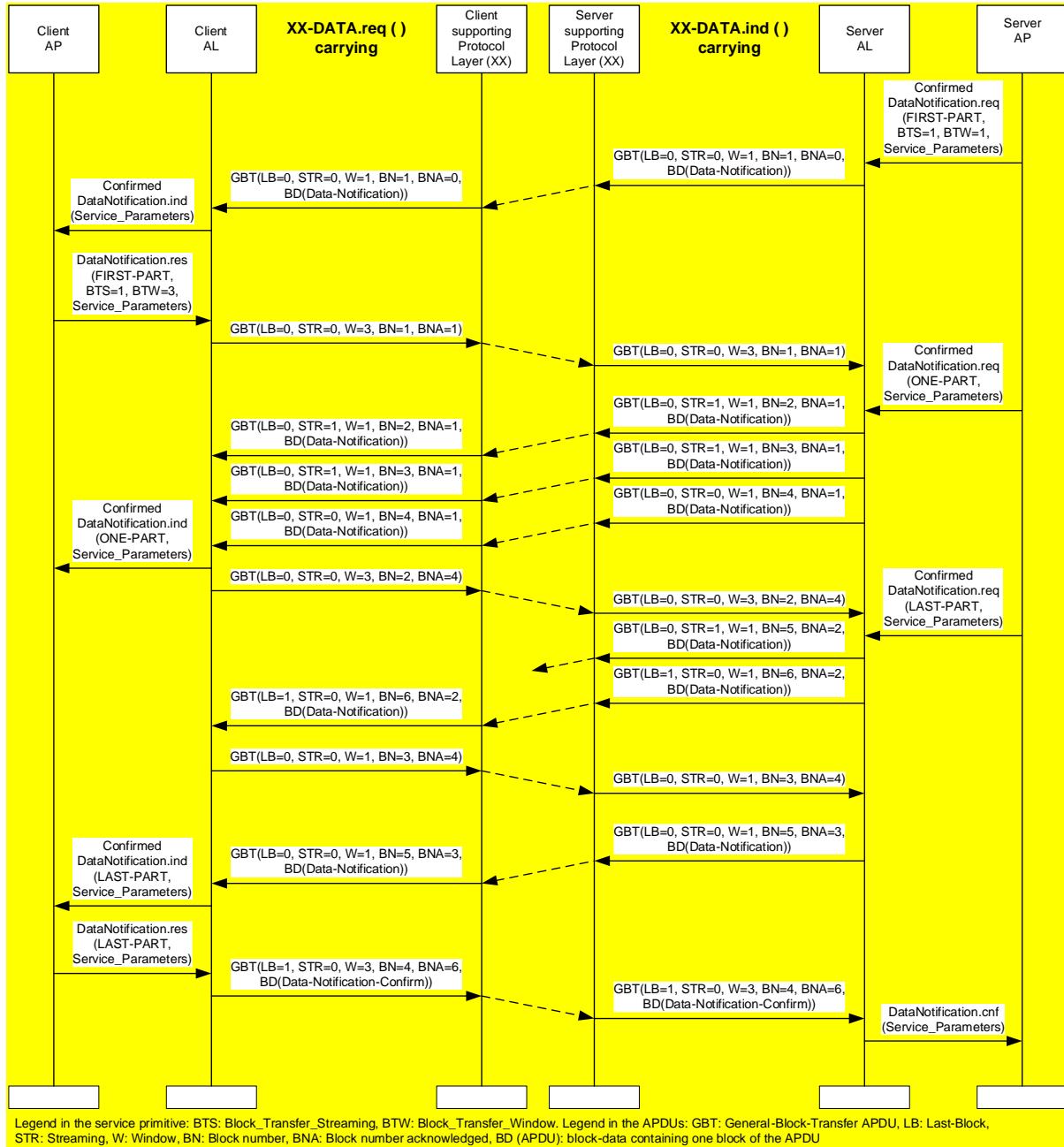


Figure 152 – DataNotification\_Confirmed with GBT recovery

Figure 152 shows transporting a Confirmed DataNotification service with GBT and with list block recovery.

The process is essentially the same as shown in Figure 126, but the 5<sup>th</sup> block of the server is lost.

The client AI detects this and ask the server to resend it by sending a GBT APDU with LB = 0, STR = 0, W = 1, BN = 3 and BNA = 4.

The process continues then normally.

#### 9.4.6.13.8 Aborting the GBT procedure

A GBT procedure can be aborted any time by:

- an ABORT GBT APDU with LB = 1, STR = FALSE, BN = 0 and BNA = 0;
- starting a new GBT procedure. This is detected by receiving a GBT APDU with BN = 1 and BNA = 0;
- an APDU other than GBT.

A confirmed GBT procedure shall also be aborted if a party confirms the reception of a block not yet sent by the other party.

It is not possible to abort GBT with unconfirmed DataNotification.

#### 9.4.6.14 Protocol of exception mechanism

xDLMS service requests may result in an exception when the request cannot be processed and therefore, the response corresponding to the request cannot be sent back. In such cases, the server optionally uses a confirmedServiceError APDU in case of SN referencing and exception-response APDU in case of LN referencing.

- confirmedServiceError APDU: this is an APDU that can be sent by the server to provide service-specific diagnostic information if a confirmed service .request invocation fails and the corresponding .response primitive cannot be sent back. It can be used with the InitiateRequest, Read and Write services;
- the exception-response APDU: this is an APDU that can be sent by the server when a .response primitive corresponding to a .request using LN referencing cannot be sent back. It can be used with the GET, SET, ACTION and ACCESS services;

Conditions to use the xDLMS exception mechanism are specified in Table 97

**Table 97 – xDLMS exception mechanism**

Request processing condition	LN referencing response APDU	SN referencing response APDU
Association is not established	exception-response (state-error = service-not-allowed, service-error = operation-not-possible)	confirmedServiceError (read or write, vde-state-error = no-dlms-context)
Request APDU size > server-max-receive-pdu-size	exception-response (state-error = service-not-allowed, service-error = pdu-too-long)	confirmedServiceError (read or write, service = pdu-size)
Protection is not verified or removed due to invocation counter error	exception-response (state-error = service-not-allowed, service-error = invocation-counter-error)  value = lowest acceptable value of invocation counter	confirmedServiceError (read or write, application-reference = deciphering-error)
Protection is not verified or removed due to deciphering error	exception-response (state-error = service-not-allowed, service-error = deciphering-error)	confirmedServiceError (read or write, application-reference = deciphering-error)
Request is not well formed	exception-response (state-error = service-not-known, service-error = operation-not-possible)	confirmedServiceError (read or write, definition = other)
Request not supported with negotiated conformance	exception-response (state-error = service-not-allowed, service-error = service-not-supported)	confirmedServiceError (read or write, service = service-unsupported)

## 9.5 Abstract syntax of COSEM PDUs

The abstract syntax of COSEM PDUs is specified in this subclause using ASN.1. See ISO/IEC 8824:2008.

NOTE The CIASE APDUs are specified in 10.5.9.

```

COSEMPdu DEFINITIONS ::= BEGIN

ACSE-APDU ::= CHOICE
{
  aarq                      AARQ-apdu,
  aare                      AARE-apdu,
  rlrq                      RLRQ-apdu,           -- OPTIONAL
  rlre                      RLRE-apdu           -- OPTIONAL
}

XDLMS-APDU ::= CHOICE
{
-- standardised xDLMS pdus used in DLMS/COSEM

-- with no ciphering

  initiateRequest            [1] IMPLICIT    InitiateRequest,
  readRequest                [5] IMPLICIT    ReadRequest,
  writeRequest               [6] IMPLICIT    WriteRequest,

  initiateResponse           [8] IMPLICIT    InitiateResponse,
  readResponse               [12] IMPLICIT   ReadResponse,
  writeResponse              [13] IMPLICIT   WriteResponse,

  confirmedServiceError      [14]           ConfirmedServiceError,

-- data-notification

  data-notification          [15] IMPLICIT    Data-Notification,
  data-notification-confirm  [16] IMPLICIT    Data-Notification-Confirm,

  unconfirmedWriteRequest    [22] IMPLICIT    UnconfirmedWriteRequest,
  informationReportRequest   [24] IMPLICIT    InformationReportRequest,

-- The APDU tag of each ciphered xDLMS APDU indicates the type of the unciphered APDU and whether
-- global or dedicated key is used. The type of the key is carried by the security header, and
-- after removing the encryption and/or verifying the authentication tag, the original APDU with
-- its APDU TAG is restored. Therefore, the APDU tags of the ciphered APDUs carry redundant
-- information, but they are retained for consistency.

-- with global ciphering

  glo-initiateRequest        [33] IMPLICIT    OCTET STRING,
  glo-readRequest            [37] IMPLICIT    OCTET STRING,
  glo-writeRequest           [38] IMPLICIT    OCTET STRING,

  glo-initiateResponse       [40] IMPLICIT    OCTET STRING,
  glo-readResponse           [44] IMPLICIT    OCTET STRING,
  glo-writeResponse          [45] IMPLICIT    OCTET STRING,

  glo-confirmedServiceError  [46] IMPLICIT    OCTET STRING,
  glo-unconfirmedWriteRequest [54] IMPLICIT    OCTET STRING,
  glo-informationReportRequest [56] IMPLICIT    OCTET STRING,

-- with dedicated ciphering

  ded-initiateRequest        [65] IMPLICIT    OCTET STRING,
  ded-readRequest             [69] IMPLICIT    OCTET STRING,
  ded-writeRequest            [70] IMPLICIT    OCTET STRING,

  ded-initiateResponse       [72] IMPLICIT    OCTET STRING,
  ded-readResponse            [76] IMPLICIT    OCTET STRING,
  ded-writeResponse           [77] IMPLICIT    OCTET STRING,

  ded-confirmedServiceError  [78] IMPLICIT    OCTET STRING,
  ded-unconfirmedWriteRequest [86] IMPLICIT    OCTET STRING,
  ded-informationReportRequest [88] IMPLICIT    OCTET STRING,

-- xDLMS APDUs used with LN referencing

```

```

-- with no ciphering

get-request [192] IMPLICIT Get-Request,
set-request [193] IMPLICIT Set-Request,
event-notification-request [194] IMPLICIT EventNotificationRequest,
action-request [195] IMPLICIT Action-Request,

get-response [196] IMPLICIT Get-Response,
set-response [197] IMPLICIT Set-Response,
action-response [199] IMPLICIT Action-Response,

-- with global ciphering

glo-get-request [200] IMPLICIT OCTET STRING,
glo-set-request [201] IMPLICIT OCTET STRING,
glo-event-notification-request [202] IMPLICIT OCTET STRING,
glo-action-request [203] IMPLICIT OCTET STRING,

glo-get-response [204] IMPLICIT OCTET STRING,
glo-set-response [205] IMPLICIT OCTET STRING,
glo-action-response [207] IMPLICIT OCTET STRING,

-- with dedicated ciphering

ded-get-request [208] IMPLICIT OCTET STRING,
ded-set-request [209] IMPLICIT OCTET STRING,
ded-event-notification-request [210] IMPLICIT OCTET STRING,
ded-actionRequest [211] IMPLICIT OCTET STRING,

ded-get-response [212] IMPLICIT OCTET STRING,
ded-set-response [213] IMPLICIT OCTET STRING,
ded-action-response [215] IMPLICIT OCTET STRING,

-- the exception response pdu

exception-response [216] IMPLICIT ExceptionResponse,

-- access

access-request [217] IMPLICIT Access-Request,
access-response [218] IMPLICIT Access-Response,

-- general APDUs

general-glo-ciphering [219] IMPLICIT General-Glo-Ciphering,
general-ded-ciphering [220] IMPLICIT General-Ded-Ciphering,
general-ciphering [221] IMPLICIT General-Ciphering,
general-signing [223] IMPLICIT General-Signing,
general-block-transfer [224] IMPLICIT General-Block-Transfer

-- The tags 230 and 231 are reserved for DLMS Gateway
-- reserved [230]
-- reserved [231]
}

AARQ-apdu ::= [APPLICATION 0] IMPLICIT SEQUENCE
{
-- [APPLICATION 0] == [ 60H ] = [ 96 ]

    protocol-version [0] IMPLICIT BIT STRING {version1 (0)}
    DEFAULT{version1},
    application-context-name [1] Application-context-name,
    called-AP-title [2] AP-title OPTIONAL,
    called-AE-qualifier [3] AE-qualifier OPTIONAL,
    called-AP-invocation-id [4] AP-invocation-identifier OPTIONAL,
    called-AE-invocation-id [5] AE-invocation-identifier OPTIONAL,
    calling-AP-title [6] AP-title OPTIONAL,
    calling-AE-qualifier [7] AE-qualifier OPTIONAL,
    calling-AP-invocation-id [8] AP-invocation-identifier OPTIONAL,
    calling-AE-invocation-id [9] AE-invocation-identifier OPTIONAL,

    -- The following field shall not be present if only the kernel is used.
    sender-acse-requirements [10] IMPLICIT ACSE-requirements OPTIONAL,

    -- The following field shall only be present if the authentication functional unit is selected.
    mechanism-name [11] IMPLICIT Mechanism-name OPTIONAL,

    -- The following field shall only be present if the authentication functional unit is selected.

    calling-authentication-value [12] EXPLICIT Authentication-value OPTIONAL,
    implementation-information [29] IMPLICIT Implementation-data OPTIONAL,
    user-information [30] EXPLICIT Association-information OPTIONAL
}

```

## DLMS/COSEM Architecture and Protocols

---

```

-- The user-information field shall carry an InitiateRequest APDU encoded in A-XDR, and then
-- encoding the resulting OCTET STRING in BER.

AARE-apdu ::= [APPLICATION 1] IMPLICIT SEQUENCE
{
-- [APPLICATION 1] == [ 61H ] = [ 97 ]

    protocol-version          [0] IMPLICIT      BIT STRING {version1 (0)}
                               DEFAULT {version1},
                               Application-context-name,
                               result,
                               result-source-diagnostic,
                               responding-AP-title   [4] OPTIONAL,
                               responding-AE-qualifier [5] OPTIONAL,
                               responding-AP-invocation-id [6] OPTIONAL,
                               responding-AE-invocation-id [7] OPTIONAL

-- The following field shall not be present if only the kernel is used.
    responder-acse-requirements [8] IMPLICIT      ACSE-requirements OPTIONAL,

-- The following field shall only be present if the authentication functional unit is selected.
    mechanism-name            [9] IMPLICIT      Mechanism-name OPTIONAL,

-- The following field shall only be present if the authentication functional unit is selected.
    responding-authentication-value [10] EXPLICIT Authentication-value OPTIONAL,
    implementation-information   [29] IMPLICIT      Implementation-data OPTIONAL,
    user-information             [30] EXPLICIT      Association-information OPTIONAL
}

-- The user-information field shall carry either an InitiateResponse (or, when the proposed xDLMS
-- context is not accepted by the server, a ConfirmedServiceError) APDU encoded in A-XDR, and then
-- encoding the resulting OCTET STRING in BER.

RLRQ-apdu ::= [APPLICATION 2] IMPLICIT SEQUENCE
{
-- [APPLICATION 2] == [ 62H ] = [ 98 ]

    reason                  [0] IMPLICIT      Release-request-reason OPTIONAL,
    user-information         [30] EXPLICIT      Association-information OPTIONAL
}

RLRE-apdu ::= [APPLICATION 3] IMPLICIT SEQUENCE
{
-- [APPLICATION 3] == [ 63H ] = [ 99 ]

    reason                  [0] IMPLICIT      Release-response-reason OPTIONAL,
    user-information         [30] EXPLICIT      Association-information OPTIONAL
}

-- The user-information field of the RLRQ / RLRE APDU may carry an InitiateRequest APDU encoded in
-- A-XDR, and then encoding the resulting OCTET STRING in BER, when the AA to be released uses
-- ciphering.

-- types used in the fields of the ACSE APDUs, in the order of their occurrence

Application-context-name ::= OBJECT IDENTIFIER
AP-title ::= OCTET STRING
AE-qualifier ::= OCTET STRING
AP-invocation-identifier ::= INTEGER
AE-invocation-identifier ::= INTEGER
ACSE-requirements ::= BIT STRING {authentication(0)}
Mechanism-name ::= OBJECT IDENTIFIER

Authentication-value ::= CHOICE
{
    charstring              [0] IMPLICIT      GraphicString,
    bitstring                [1] IMPLICIT      BIT STRING
}

Implementation-data ::= GraphicString
Association-information ::= OCTET STRING
Association-result ::= INTEGER
{
    accepted                (0),
    rejected-permanent     (1),
    rejected-transient     (2)
}

```

```

Associate-source-diagnostic ::= CHOICE
{
    acse-service-user           [1] INTEGER
    {
        null                      (0),
        no-reason-given          (1),
        application-context-name-not-supported (2),
        calling-AP-title-not-recognized (3),
        calling-AP-invocation-identifier-not-recognized (4),
        calling-AE-qualifier-not-recognized (5),
        calling-AE-invocation-identifier-not-recognized (6),
        called-AP-title-not-recognized (7),
        called-AP-invocation-identifier-not-recognized (8),
        called-AE-qualifier-not-recognized (9),
        called-AE-invocation-identifier-not-recognized (10),
        authentication-mechanism-name-not-recognised (11),
        authentication-mechanism-name-required (12),
        authentication-failure      (13),
        authentication-required     (14)
    },
    acse-service-provider        [2] INTEGER
    {
        null                      (0),
        no-reason-given          (1),
        no-common-acse-version   (2)
    }
}

Release-request-reason ::= INTEGER
{
    normal                     (0),
    urgent                     (1),
    user-defined               (30)
}

Release-response-reason ::= INTEGER
{
    normal                     (0),
    not-finished               (1),
    user-defined               (30)
}

-- Useful types

Integer8 ::= INTEGER(-128..127)
Integer16 ::= INTEGER(-32768..32767)
Integer32 ::= INTEGER(-2147483648..2147483647)
Integer64 ::= INTEGER(-9223372036854775808..9223372036854775807)
Unsigned8 ::= INTEGER(0..255)
Unsigned16 ::= INTEGER(0..65535)
Unsigned32 ::= INTEGER(0..4294967295)
Unsigned64 ::= INTEGER(0..18446744073709551615)

-- xDLMS APDU-s used during Association establishment

InitiateRequest ::= SEQUENCE
{
    -- shall not be encoded in DLMS without ciphering
    dedicated-key             OCTET STRING OPTIONAL,
    response-allowed          BOOLEAN DEFAULT TRUE,
    proposed-quality-of-service [0] IMPLICIT Integer8 OPTIONAL,
    proposed-dlms-version-number Unsigned8,
    proposed-conformance       Conformance, -- Shall be encoded in BER
    client-max-receive-pdu-size Unsigned16
}

-- In DLMS/COSEM, the quality-of-service parameter is not used. Any value shall be accepted.

-- The Conformance field shall be encoded in BER. See IEC 61334-6 Example 1.

InitiateResponse ::= SEQUENCE
{
    negotiated-quality-of-service [0] IMPLICIT Integer8 OPTIONAL,
    negotiated-dlms-version-number Unsigned8,
    negotiated-conformance       Conformance, -- Shall be encoded in BER
    server-max-receive-pdu-size Unsigned16,
    vaa-name                     ObjectName
}

-- In the case of LN referencing, the value of the vaa-name is 0x0007
-- In the case of SN referencing, the value of the vaa-name is the base name of the
-- Current Association object, 0xF0A0
-- Conformance Block

-- SIZE constrained BIT STRING is extension of ASN.1 notation

```

```

Conformance ::= [APPLICATION 31] IMPLICIT BIT STRING
{
    -- the bit is set when the corresponding service or functionality is available
    reserved-zero          (0),
    -- The actual list of general protection services depends on the security suite
    general-protection     (1),
    general-block-transfer (2),
    read                   (3),
    write                  (4),
    unconfirmed-write      (5),
    delta-value-encoding   (6),
    reserved-seven         (7),
    attribute0-supported-with-set (8),
    priority-mgmt-supported (9),
    attribute0-supported-with-get (10),
    block-transfer-with-get-or-read (11),
    block-transfer-with-set-or-write (12),
    block-transfer-with-action (13),
    multiple-references   (14),
    information-report    (15),
    data-notification     (16),
    access                 (17),
    parameterized-access  (18),
    get                   (19),
    set                   (20),
    selective-access       (21),
    event-notification    (22),
    action                (23)
}

ObjectName ::= Integer16
-- for named variable objects (short names), the last three bits shall be set to 000;
-- for vaa-name objects, the last three bits shall be set to 111.

-- The Confirmed ServiceError APDU is used only with the InitiateRequest, ReadRequest and
-- WriteRequest APDUs when the request fails, to provide diagnostic information.

ConfirmedServiceError ::= CHOICE
{
-- tag 0 is reserved
-- In DLMS/COSEM only initiateError, read and write are relevant

    initiateError           [1] ServiceError,
    getStatus               [2] ServiceError,
    getNameList             [3] ServiceError,
    getVariableAttribute   [4] ServiceError,
    read                    [5] ServiceError,
    write                   [6] ServiceError,
    getDataSetAttribute    [7] ServiceError,
    getTIAAttribute         [8] ServiceError,
    changeScope             [9] ServiceError,
    start                  [10] ServiceError,
    stop                   [11] ServiceError,
    resume                 [12] ServiceError,
    makeUsable              [13] ServiceError,
    initiateLoad            [14] ServiceError,
    loadSegment              [15] ServiceError,
    terminateLoad            [16] ServiceError,
    initiateUpLoad          [17] ServiceError,
    upLoadSegment            [18] ServiceError,
    terminateUpLoad          [19] ServiceError
}

ServiceError ::= CHOICE
{
    application-reference [0] IMPLICIT ENUMERATED
    {
        -- DLMS provider only
        other                      (0),
        time-elapsed               (1), -- time out since request sent
        application-unreachable   (2), -- peer AEi not reachable
        application-reference-invalid (3), -- addressing trouble
        application-context-unsupported (4), -- application-context incompatibility
        provider-communication-error (5), -- error at the local or distant equipment
        deciphering-error          (6) -- error detected by the deciphering function
    },
    hardware-resource [1] IMPLICIT ENUMERATED
    {
        -- VDE hardware troubles
        other                      (0),
        memory-unavailable        (1),
        processor-resource-unavailable (2),
        mass-storage-unavailable  (3),
        other-resource-unavailable (4)
    }
}

```

```

vde-state-error          [2] IMPLICIT ENUMERATED
{
-- Error source description
    other                  (0),
    no-dlms-context       (1),
    loading-data-set      (2),
    status-nochange        (3),
    status-inoperable     (4)
},
service                 [3] IMPLICIT ENUMERATED
{
-- service handling troubles
    other                  (0),
    pdu-size               (1), -- pdu too long
    service-unsupported    (2)  -- as defined in the conformance block
},
definition              [4] IMPLICIT ENUMERATED
{
-- object bound troubles in a service
    other                  (0),
    object-undefined       (1), -- object not defined at the VDE
    object-class-inconsistent (2), -- class of object incompatible with asked
                                     service
    object-attribute-inconsistent (3) -- object attributes are inconsistent
},
access                  [5] IMPLICIT ENUMERATED
{
-- object access error
    other                  (0),
    scope-of-access-violated (1), -- access denied through authorisation reason
    object-access-violated (2), -- access incompatible with object attribute
    hardware-fault         (3), -- access fail for hardware reason
    object-unavailable      (4) -- VDE hands object for unavailable
},
initiate                [6] IMPLICIT ENUMERATED
{
-- initiate service error
    other                  (0),
    dlms-version-too-low   (1), -- proposed DLMS version too low
    incompatible-conformance (2), -- proposed service not sufficient
    pdu-size-too-short     (3), -- proposed PDU size too short
    refused-by-the-VDE-Handler (4) -- vaa creation impossible or not allowed
},
load-data-set           [7] IMPLICIT ENUMERATED
{
-- data set load services error
    other                  (0),
    primitive-out-of-sequence (1), -- according to the DataSet loading state
                                     transitions
    not-loadable           (2), -- loadable attribute set to FALSE
    dataset-size-too-large (3), -- evaluated Data Set size too large
    not-awaited-segment    (4), -- proposed segment not awaited
    interpretation-failure (5), -- segment interpretation error
    storage-failure        (6), -- segment storage error
    data-set-not-ready     (7) -- Data Set not in correct state for uploading
},
-- change-scope          [8] IMPLICIT ENUMERATED
task                    [9] IMPLICIT ENUMERATED
{
-- TI services error
    other                  (0),
    no-remote-control     (1), -- Remote Control parameter set to FALSE
    ti-stopped             (2), -- TI in stopped state
    ti-running             (3), -- TI in running state
    ti-unusable            (4) -- TI in unusable state
},
-- other                  [10] IMPLICIT ENUMERATED
}

```

```
-- COSEM APDUs using short name referencing

ReadRequest ::= SEQUENCE OF Variable-Access-Specification

ReadResponse ::= SEQUENCE OF CHOICE
{
    data [0] Data,
    data-access-error [1] IMPLICIT Data-Access-Result,
    data-block-result [2] IMPLICIT Data-Block-Result,
    block-number [3] IMPLICIT Unsigned16
}

WriteRequest ::= SEQUENCE
{
    variable-access-specification SEQUENCE OF Variable-Access-Specification,
    list-of-data SEQUENCE OF Data
}

WriteResponse ::= SEQUENCE OF CHOICE
{
    success [0] IMPLICIT NULL,
    data-access-error [1] IMPLICIT Data-Access-Result,
    block-number [2] Unsigned16
}

UnconfirmedWriteRequest ::= SEQUENCE
{
    variable-access-specification SEQUENCE OF Variable-Access-Specification,
    list-of-data SEQUENCE OF Data
}

InformationReportRequest ::= SEQUENCE
{
    current-time GeneralizedTime OPTIONAL,
    variable-access-specification SEQUENCE OF Variable-Access-Specification,
    list-of-data SEQUENCE OF Data
}

-- COSEM APDUs using logical name referencing

Get-Request ::= CHOICE
{
    get-request-normal [1] IMPLICIT Get-Request-Normal,
    get-request-next [2] IMPLICIT Get-Request-Next,
    get-request-with-list [3] IMPLICIT Get-Request-With-List
}

Get-Request-Normal ::= SEQUENCE
{
    invoke-id-and-priority Invoke-Id-And-Priority,
    cosem-attribute-descriptor Cosem-Attribute-Descriptor,
    access-selection Selective-Access-Descriptor OPTIONAL
}

Get-Request-Next ::= SEQUENCE
{
    invoke-id-and-priority Invoke-Id-And-Priority,
    block-number Unsigned32
}

Get-Request-With-List ::= SEQUENCE
{
    invoke-id-and-priority Invoke-Id-And-Priority,
    attribute-descriptor-list SEQUENCE OF Cosem-Attribute-Descriptor-With-Selection
}

Get-Response ::= CHOICE
{
    get-response-normal [1] IMPLICIT Get-Response-Normal,
    get-response-with-datablock [2] IMPLICIT Get-Response-With-Datablock,
    get-response-with-list [3] IMPLICIT Get-Response-With-List
}

Get-Response-Normal ::= SEQUENCE
{
    invoke-id-and-priority Invoke-Id-And-Priority,
    result Get-Data-Result
}

Get-Response-With-Datablock ::= SEQUENCE
{
    invoke-id-and-priority Invoke-Id-And-Priority,
    result DataBlock-G
}

Get-Response-With-List ::= SEQUENCE
{
```

```

    invoke-id-and-priority           Invoke-Id-And-Priority,
    result                           SEQUENCE OF Get-Data-Result
}

Set-Request ::= CHOICE
{
    set-request-normal             [1] IMPLICIT Set-Request-Normal,
    set-request-with-first-datablock [2] IMPLICIT Set-Request-With-First-Datablock,
    set-request-with-datablock      [3] IMPLICIT Set-Request-With-Datablock,
    set-request-with-list          [4] IMPLICIT Set-Request-With-List,
    set-request-with-list-and-first-datablock [5] IMPLICIT Set-Request-With-List-And-First-Datablock
}

Set-Request-Normal ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    cosem-attribute-descriptor      Cosem-Attribute-Descriptor,
    access-selection                 Selective-Access-Descriptor OPTIONAL,
    value                            Data
}

Set-Request-With-First-Datablock ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    cosem-attribute-descriptor      Cosem-Attribute-Descriptor,
    access-selection                 [0] IMPLICIT Selective-Access-Descriptor OPTIONAL,
    datablock                         DataBlock-SA
}

Set-Request-With-Datablock ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    datablock                         DataBlock-SA
}

Set-Request-With-List ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    attribute-descriptor-list       SEQUENCE OF Cosem-Attribute-Descriptor-With-Selection,
    value-list                        SEQUENCE OF Data
}

Set-Request-With-List-And-First-Datablock ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    attribute-descriptor-list       SEQUENCE OF Cosem-Attribute-Descriptor-With-Selection,
    datablock                         DataBlock-SA
}

Set-Response ::= CHOICE
{
    set-response-normal             [1] IMPLICIT Set-Response-Normal,
    set-response-datablock          [2] IMPLICIT Set-Response-Datablock,
    set-response-last-datablock     [3] IMPLICIT Set-Response-Last-Datablock,
    set-response-last-datablock-with-list [4] IMPLICIT Set-Response-Last-Datablock-With-List,
    set-response-with-list          [5] IMPLICIT Set-Response-With-List
}

Set-Response-Normal ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    result                           Data-Access-Result
}

Set-Response-Datablock ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    block-number                     Unsigned32
}

Set-Response-Last-Datablock ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    result                           Data-Access-Result,
    block-number                     Unsigned32
}

Set-Response-Last-Datablock-With-List ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
    result                           SEQUENCE OF Data-Access-Result,
    block-number                     Unsigned32
}

Set-Response-With-List ::= SEQUENCE
{
    invoke-id-and-priority           Invoke-Id-And-Priority,
}

```

```

        result                               SEQUENCE OF Data-Access-Result
}

Action-Request ::= CHOICE
{
    action-request-normal                [1] IMPLICIT Action-Request-Normal,
    action-request-next-pblock           [2] IMPLICIT Action-Request-Next-Pblock,
    action-request-with-list             [3] IMPLICIT Action-Request-With-List,
    action-request-with-first-pblock     [4] IMPLICIT Action-Request-With-First-Pblock,
    action-request-with-list-and-first-pblock [5] IMPLICIT Action-Request-With-List-And-First-Pblock,
    action-request-with-pblock           [6] IMPLICIT Action-Request-With-Pblock
}

Action-Request-Normal ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    cosem-method-descriptor            Cosem-Method-Descriptor,
    method-invocation-parameters      Data OPTIONAL
}

Action-Request-Next-Pblock ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    block-number                         Unsigned32
}

Action-Request-With-List ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    cosem-method-descriptor-list       SEQUENCE OF Cosem-Method-Descriptor,
    method-invocation-parameters      SEQUENCE OF Data
}

Action-Request-With-First-Pblock ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    cosem-method-descriptor            Cosem-Method-Descriptor,
    pblock                             DataBlock-SA
}

Action-Request-With-List-And-First-Pblock ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    cosem-method-descriptor-list       SEQUENCE OF Cosem-Method-Descriptor,
    pblock                             DataBlock-SA
}

Action-Request-With-Pblock ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    pblock                             DataBlock-SA
}

Action-Response ::= CHOICE
{
    action-response-normal              [1] IMPLICIT Action-Response-Normal,
    action-response-with-pblock         [2] IMPLICIT Action-Response-With-Pblock,
    action-response-with-list           [3] IMPLICIT Action-Response-With-List,
    action-response-next-pblock         [4] IMPLICIT Action-Response-Next-Pblock
}

Action-Response-Normal ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    single-response                    Action-Response-With-Optional-Data
}

Action-Response-With-Pblock ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    pblock                             DataBlock-SA
}

Action-Response-With-List ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    list-of-responses                  SEQUENCE OF Action-Response-With-Optional-Data
}

Action-Response-Next-Pblock ::= SEQUENCE
{
    invoke-id-and-priority              Invoke-Id-And-Priority,
    block-number                         Unsigned32
}

EventNotificationRequest ::= SEQUENCE
{
}

```

```

time                                OCTET STRING OPTIONAL,
cosem-attribute-descriptor          Cosem-Attribute-Descriptor,
attribute-value                      Data
}

ExceptionResponse ::= SEQUENCE
{
    state-error                  [0] IMPLICIT ENUMERATED
    {
        service-not-allowed      (1),
        service-unknown          (2)
    },
    service-error                [1] CHOICE
    {
        operation-not-possible   [1] IMPLICIT NULL,
        service-not-supported    [2] IMPLICIT NULL,
        other-reason              [3] IMPLICIT NULL,
        pdu-too-long              [4] IMPLICIT NULL,
        deciphering-error         [5] IMPLICIT NULL,
        invocation-counter-error [6] IMPLICIT Unsigned32
    }
}

-- Access

Access-Request ::= SEQUENCE
{
    long-invoke-id-and-priority      Long-Invoke-Id-And-Priority,
    date-time                         OCTET STRING,
    access-request-body               Access-Request-Body
}

Access-Response ::= SEQUENCE
{
    long-invoke-id-and-priority      Long-Invoke-Id-And-Priority,
    date-time                         OCTET STRING,
    access-response-body             Access-Response-Body
}

-- Data-Notification

Data-Notification ::= SEQUENCE
{
    long-invoke-id-and-priority      Long-Invoke-Id-And-Priority,
    date-time                         OCTET STRING,
    notification-body                Notification-Body
}

Data-Notification-Confirm ::= SEQUENCE
{
    long-invoke-id-and-priority      Long-Invoke-Id-And-Priority,
    date-time                         OCTET STRING
}

-- General APDUs

General-Ded-Ciphering ::= SEQUENCE
{
    system-title                    OCTET STRING,
    ciphered-content                OCTET STRING
}

General-Glo-Ciphering ::= SEQUENCE
{
    system-title                    OCTET STRING,
    ciphered-content                OCTET STRING
}

General-Ciphering ::= SEQUENCE
{
    transaction-id                 OCTET STRING,
    originator-system-title        OCTET STRING,
    recipient-system-title         OCTET STRING,
    date-time                      OCTET STRING,
    other-information               OCTET STRING,
    key-info OPTIONAL,             Key-Info OPTIONAL,
    ciphered-content                OCTET STRING
}

```

```

General-Signing ::= SEQUENCE
{
    transaction-id          OCTET STRING,
    originator-system-title OCTET STRING,
    recipient-system-title  OCTET STRING,
    date-time               OCTET STRING,
    other-information        OCTET STRING,
    content                 OCTET STRING,
    signature               OCTET STRING
}

General-Block-Transfer ::= SEQUENCE
{
    block-control           Block-Control,
    block-number             Unsigned16,
    block-number-ack         Unsigned16,
    block-data               OCTET STRING
}

-- Types used in the xDLMS data transfer services

Variable-Access-Specification ::= CHOICE
{
    variable-name            [2] IMPLICIT ObjectName,
-- detailed-access [3] is not used in DLMS/COSEM
    parameterized-access      [4] IMPLICIT Parameterized-Access,
    block-number-access       [5] IMPLICIT Block-Number-Access,
    read-data-block-access   [6] IMPLICIT Read-Data-Block-Access,
    write-data-block-access  [7] IMPLICIT Write-Data-Block-Access
}

Parameterized-Access ::= SEQUENCE
{
    variable-name            ObjectName,
    selector                 Unsigned8,
    parameter                Data
}

Block-Number-Access ::= SEQUENCE
{
    block-number              Unsigned16
}

Read-Data-Block-Access ::= SEQUENCE
{
    last-block                BOOLEAN,
    block-number               Unsigned16,
    raw-data                  OCTET STRING
}

Write-Data-Block-Access ::= SEQUENCE
{
    last-block                BOOLEAN,
    block-number               Unsigned16
}

Data ::= CHOICE
{
    null-data                 [0] IMPLICIT NULL,
    array                     [1] IMPLICIT SEQUENCE OF Data,
    structure                 [2] IMPLICIT SEQUENCE OF Data,
    boolean                   [3] IMPLICIT BOOLEAN,
    bit-string                [4] IMPLICIT BIT STRING,
    double-long               [5] IMPLICIT Integer32,
    double-long-unsigned      [6] IMPLICIT Unsigned32,
    octet-string              [9] IMPLICIT OCTET STRING,
    visible-string            [10] IMPLICIT VisibleString,
    utf8-string               [12] IMPLICIT UTF8String,
    bcd                       [13] IMPLICIT Integer8,
    integer                   [15] IMPLICIT Integer8,
    long                      [16] IMPLICIT Integer16,
    unsigned                  [17] IMPLICIT Unsigned8,
    long-unsigned              [18] IMPLICIT Unsigned16,
    compact-array              [19] IMPLICIT SEQUENCE
    {
        contents-description [0] TypeDescription,
        array-contents       [1] IMPLICIT OCTET STRING
    },
    long64                    [20] IMPLICIT Integer64,
    long64-unsigned            [21] IMPLICIT Unsigned64,
    enum                      [22] IMPLICIT Unsigned8,
    float32                  [23] IMPLICIT OCTET STRING (SIZE(4)),
    float64                  [24] IMPLICIT OCTET STRING (SIZE(8)),
    date-time                 [25] IMPLICIT OCTET STRING (SIZE(12)),
    date                      [26] IMPLICIT OCTET STRING (SIZE(5)),
}

```

```

time                               [27] IMPLICIT OCTET STRING (SIZE(4)),
delta-integer                      [28] IMPLICIT Integer8,
delta-long                          [29] IMPLICIT Integer16,
delta-double-long                  [30] IMPLICIT Integer32,
delta-unsigned                      [31] IMPLICIT Unsigned8,
delta-long-unsigned                [32] IMPLICIT Unsigned16,
delta-double-long-unsigned          [33] IMPLICIT Unsigned32,
dont-care                           [255] IMPLICIT NULL
}

-- The following TypeDescription relates to the compact-array data Type

TypeDescription ::= CHOICE
{
    null-data                         [0] IMPLICIT NULL,
    array                             [1] IMPLICIT SEQUENCE
    {
        number-of-elements           Unsigned16,
        type-description            TypeDescription
    },
    structure                          [2] IMPLICIT SEQUENCE OF TypeDescription,
    boolean                            [3] IMPLICIT NULL,
    bit-string                         [4] IMPLICIT NULL,
    double-long                        [5] IMPLICIT NULL,
    double-long-unsigned              [6] IMPLICIT NULL,
    octet-string                      [9] IMPLICIT NULL,
    visible-string                    [10] IMPLICIT NULL,
    utf8-string                       [12] IMPLICIT NULL,
    bcd                                [13] IMPLICIT NULL,
    integer                            [15] IMPLICIT NULL,
    long                               [16] IMPLICIT NULL,
    unsigned                           [17] IMPLICIT NULL,
    long-unsigned                     [18] IMPLICIT NULL,
    long64                             [20] IMPLICIT NULL,
    long64-unsigned                   [21] IMPLICIT NULL,
    enum                               [22] IMPLICIT NULL,
    float32                            [23] IMPLICIT NULL,
    float64                            [24] IMPLICIT NULL,
    date-time                          [25] IMPLICIT NULL,
    date                               [26] IMPLICIT NULL,
    time                               [27] IMPLICIT NULL,
    dont-care                          [255] IMPLICIT NULL
}

Data-Access-Result ::= ENUMERATED
{
    success                           (0),
    hardware-fault                   (1),
    temporary-failure               (2),
    read-write-denied                (3),
    object-undefined                 (4),
    object-class-inconsistent       (9),
    object-unavailable              (11),
    type-unmatched                  (12),
    scope-of-access-violated        (13),
    data-block-unavailable          (14),
    long-get-aborted                (15),
    no-long-get-in-progress         (16),
    long-set-aborted                (17),
    no-long-set-in-progress         (18),
    data-block-number-invalid       (19),
    other-reason                     (250)
}

Action-Result ::= ENUMERATED
{
    success                           (0),
    hardware-fault                   (1),
    temporary-failure               (2),
    read-write-denied                (3),
    object-undefined                 (4),
    object-class-inconsistent       (9),
    object-unavailable              (11),
    type-unmatched                  (12),
    scope-of-access-violated        (13),
    data-block-unavailable          (14),
    long-action-aborted             (15),
    no-long-action-in-progress      (16),
    data-block-number-invalid       (19),
    other-reason                     (250)
}

-- IEC 61334-6 clause 5 specifies that bits of any byte are numbered from 1 to 8,
-- where bit 8 is the most significant.
-- In the DLMS UA Green Book, bits are numbered from 0 to 7.
-- Use of Invoke-Id-And-Priority
--   invoke-id                      bits 0-3

```

```

-- reserved bits 4-5
-- service-class bit 6 0 = Unconfirmed, 1 = Confirmed
-- priority bit 7 0 = Normal, 1 = High
Invoke-Id-And-Priority ::= Unsigned8

-- Use of Long-Invoke-Id-And-Priority
-- long-invoke-id bits 0-23
-- reserved bits 24-27
-- self-descriptive bit 28 0 = Not-Self-Descriptive, 1 = Self-Descriptive
-- processing-option bit 29 0 = Continue on Error, 1 = Break on Error
-- service-class bit 30 0 = Unconfirmed, 1 = Confirmed
-- priority bit 31 0 = Normal, 1 = High
Long-Invoke-Id-And-Priority ::= Unsigned32

Cosem-Attribute-Descriptor ::= SEQUENCE
{
    class-id Cosem-Class-Id,
    instance-id Cosem-Object-Instance-Id,
    attribute-id Cosem-Object-Attribute-Id
}

Cosem-Method-Descriptor ::= SEQUENCE
{
    class-id Cosem-Class-Id,
    instance-id Cosem-Object-Instance-Id,
    method-id Cosem-Object-Method-Id
}

Cosem-Class-Id ::= Unsigned16

Cosem-Object-Instance-Id ::= OCTET STRING (SIZE(6))

Cosem-Object-Attribute-Id ::= Integer8

Cosem-Object-Method-Id ::= Integer8

Selective-Access-Descriptor ::= SEQUENCE
{
    access-selector Unsigned8,
    access-parameters Data
}

Cosem-Attribute-Descriptor-With-Selection ::= SEQUENCE
{
    cosem-attribute-descriptor Cosem-Attribute-Descriptor,
    access-selection Selective-Access-Descriptor OPTIONAL
}

Get-Data-Result ::= CHOICE
{
    data [0] Data,
    data-access-result [1] IMPLICIT Data-Access-Result
}

Data-Block-Result ::= SEQUENCE -- Used in ReadResponse with block transfer
{
    last-block BOOLEAN,
    block-number Unsigned16,
    raw-data OCTET STRING
}

DataBlock-G ::= SEQUENCE -- G == DataBlock for the GET-response
{
    last-block BOOLEAN,
    block-number Unsigned32,
    result CHOICE
    {
        raw-data [0] IMPLICIT OCTET STRING,
        data-access-result [1] IMPLICIT Data-Access-Result
    }
}

DataBlock-SA ::= SEQUENCE -- SA == DataBlock for the SET-request, ACTION-request and ACTION-response
{
    last-block BOOLEAN,
    block-number Unsigned32,
    raw-data OCTET STRING
}

Action-Response-With-Optional-Data ::= SEQUENCE
{
    result Action-Result,
    return-parameters Get-Data-Result OPTIONAL
}

Notification-Body ::= SEQUENCE
{
}

```

```

    data-value                               Data
}

List-Of-Data ::= SEQUENCE OF Data

Access-Request-Get ::= SEQUENCE
{
    cosem-attribute-descriptor      Cosem-Attribute-Descriptor
}

Access-Request-Get-With-Selection ::= SEQUENCE
{
    cosem-attribute-descriptor      Cosem-Attribute-Descriptor,
    access-selection                Selective-Access-Descriptor
}

Access-Request-Set ::= SEQUENCE
{
    cosem-attribute-descriptor      Cosem-Attribute-Descriptor
}

Access-Request-Set-With-Selection ::= SEQUENCE
{
    cosem-attribute-descriptor      Cosem-Attribute-Descriptor,
    access-selection                Selective-Access-Descriptor
}

Access-Request-Action ::= SEQUENCE
{
    cosem-method-descriptor        Cosem-Method-Descriptor
}

Access-Request-Specification ::= CHOICE
{
    access-request-get            [1] Access-Request-Get,
    access-request-set            [2] Access-Request-Set,
    access-request-action         [3] Access-Request-Action,
    access-request-get-with-selection [4] Access-Request-Get-With-Selection,
    access-request-set-with-selection [5] Access-Request-Set-With-Selection
}

List-Of-Access-Request-Specification ::= SEQUENCE OF Access-Request-Specification

Access-Request-Body ::= SEQUENCE
{
    access-request-specification   List-Of-Access-Request-Specification,
    access-request-list-of-data    List-Of-Data
}

Access-Response-Get ::= SEQUENCE
{
    result                         Data-Access-Result
}

Access-Response-Set ::= SEQUENCE
{
    result                         Data-Access-Result
}

Access-Response-Action ::= SEQUENCE
{
    result                         Action-Result
}

Access-Response-Specification ::= CHOICE
{
    access-response-get           [1] Access-Response-Get,
    access-response-set           [2] Access-Response-Set,
    access-response-action        [3] Access-Response-Action
}

List-Of-Access-Response-Specification ::= SEQUENCE OF Access-Response-Specification

Access-Response-Body ::= SEQUENCE
{
    access-request-specification   [0] List-Of-Access-Request-Specification OPTIONAL,
    access-response-list-of-data   List-Of-Data,
    access-response-specification  List-Of-Access-Response-Specification
}

```

```

-- Key-info

Key-Id ::= ENUMERATED
{
    global-unicast-encryption-key      (0),
    global-broadcast-encryption-key    (1)
}

Kek-Id ::= ENUMERATED
{
    master-key                      (0)
}

Identified-Key ::= SEQUENCE
{
    key-id                         Key-Id
}

Wrapped-Key ::= SEQUENCE
{
    kek-id,                         Kek-Id,
    key-ciphered-data               OCTET STRING
}

Agreed-Key ::= SEQUENCE
{
    key-parameters                 OCTET STRING,
    key-ciphered-data               OCTET STRING
}

Key-Info ::= CHOICE
{
    identified-key                  [0] Identified-Key,
    wrapped-key                     [1] Wrapped-Key,
    agreed-key                      [2] Agreed-Key
}

-- Use of Block-Control
-- window                         bits 0-5      window advertise
-- streaming                       bit 6        0 = No Streaming active, 1 = Streaming active
-- last-block                      bit 7        0 = Not Last Block, 1 = Last Block
Block-Control ::= Unsigned8

```

END

## 9.6 COSEM PDU XML schema

### 9.6.1 General

ITU-T recommendations X.693 and X.694 provide XML encoding rules to Abstract Syntax Notation 1 (ASN.1) and XML Schema Definitions Language (XSD) mapping to ASN.1. No recommendation is provided to map ASN.1 to XSD.

XML has gained wide acceptance in the IT industry. The purpose of such encoding is to enable transfer of COSEM model content with various means in the form of XML encoded content. It can be a XML document exchanged between applications, content included in Web services SOAP messages or content encapsulated in e-mail messages, to name just few of the applications.

Interoperability and mapping between ASN.1 encoded APDUs and XML encoded content is important in both directions. On one side ASN.1 has enabled creation of XML encoded content with support for XML Encoding Rules (See ITU-T X.693 and ITU-T X.694). On the other hand, IT industry is searching for solutions for optimal transfer of XML content with XML optimized packaging. For that purposes conversion between W3C XML Schema and ASN.1 definition is crucial.

Conversion in both directions enables proper conversion of XML content to ASN.1 encoded content and vice versa. Subclause 9.6.2 contains mapping of COSEMPdu ASN.1 definition into COSEMPdu XML Schema (XSD).

### 9.6.2 XML Schema

```

<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
              xmlns="http://www.dlms.com/COSEMPdu">
```

```

targetNamespace="http://www.dlms.com/COSEMpdu"
elementFormDefault="qualified"


<xsd:complexType name="NULL" final="#all" />

<xsd:simpleType name="BitString">
  <xsd:restriction base="xsd:string">
    <xsd:pattern value="[0-1]{0,}" />
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="ObjectIdentifier">
  <xsd:restriction base="xsd:token">
    <xsd:pattern value="[0-2]([.[1-3]?[0-9]?(\.\d+)*)?" />
  </xsd:restriction>
</xsd:simpleType>


<xsd:element name="aCSE-APDU" type="ACSE-APDU"/>
<xsd:complexType name="ACSE-APDU">
  <xsd:choice>
    <xsd:element name="aарq" type="AARQ-apdu"/>
    <xsd:element name="aарe" type="AARE-apdu"/>
    <xsd:element name="rlrq" type="RLRQ-apdu"/>
    <xsd:element name="rlre" type="RLRE-apdu"/>
  </xsd:choice>
</xsd:complexType>


<xsd:element name="xDLMS-APDU" type="XDLMS-APDU"/>
<xsd:complexType name="XDLMS-APDU">
  <xsd:choice>
    <xsd:element name="initiateRequest" type="InitiateRequest"/>
    <xsd:element name="readRequest" type="ReadRequest"/>
    <xsd:element name="writeRequest" type="WriteRequest"/>
    <xsd:element name="initiateResponse" type="InitiateResponse"/>
    <xsd:element name="readResponse" type="ReadResponse"/>
    <xsd:element name="writeResponse" type="WriteResponse"/>
    <xsd:element name="confirmedServiceError" type="ConfirmedServiceError"/>
    <xsd:element name="data-notification" type="Data-Notification"/>
    <xsd:element name="data-notification-confirm" type="Data-Notification-Confirm"/>
    <xsd:element name="unconfirmedWriteRequest" type="UnconfirmedWriteRequest"/>
    <xsd:element name="informationReportRequest" type="InformationReportRequest"/>
    <xsd:element name="glo-initiateRequest" type="xsd:hexBinary"/>
    <xsd:element name="glo-readRequest" type="xsd:hexBinary"/>
    <xsd:element name="glo-writeRequest" type="xsd:hexBinary"/>
    <xsd:element name="glo-initiateResponse" type="xsd:hexBinary"/>
    <xsd:element name="glo-readResponse" type="xsd:hexBinary"/>
    <xsd:element name="glo-writeResponse" type="xsd:hexBinary"/>
    <xsd:element name="glo-confirmedServiceError" type="xsd:hexBinary"/>
    <xsd:element name="glo-unconfirmedWriteRequest" type="xsd:hexBinary"/>
    <xsd:element name="glo-informationReportRequest" type="xsd:hexBinary"/>
    <xsd:element name="ded-initiateRequest" type="xsd:hexBinary"/>
    <xsd:element name="ded-readRequest" type="xsd:hexBinary"/>
    <xsd:element name="ded-writeRequest" type="xsd:hexBinary"/>
    <xsd:element name="ded-initiateResponse" type="xsd:hexBinary"/>
    <xsd:element name="ded-readResponse" type="xsd:hexBinary"/>
    <xsd:element name="ded-writeResponse" type="xsd:hexBinary"/>
    <xsd:element name="ded-confirmedServiceError" type="xsd:hexBinary"/>
    <xsd:element name="ded-unconfirmedWriteRequest" type="xsd:hexBinary"/>
    <xsd:element name="ded-informationReportRequest" type="xsd:hexBinary"/>
    <xsd:element name="get-request" type="Get-Request"/>
    <xsd:element name="set-request" type="Set-Request"/>
    <xsd:element name="event-notification-request" type="EventNotificationRequest"/>
    <xsd:element name="action-request" type="Action-Request"/>
    <xsd:element name="get-response" type="Get-Response"/>
    <xsd:element name="set-response" type="Set-Response"/>
    <xsd:element name="action-response" type="Action-Response"/>
    <xsd:element name="glo-get-request" type="xsd:hexBinary"/>
    <xsd:element name="glo-set-request" type="xsd:hexBinary"/>
    <xsd:element name="glo-event-notification-request" type="xsd:hexBinary"/>
    <xsd:element name="glo-action-request" type="xsd:hexBinary"/>
  </xsd:choice>
</xsd:complexType>

```

```

t" type="xsd:hexBinary"/>
    <xsd:element name="glo-get-response" type="xsd:hexBinary"/>
    <xsd:element name="glo-set-response" type="xsd:hexBinary"/>
    <xsd:element name="glo-action-response" type="xsd:hexBinary"/>
    <xsd:element name="ded-get-request" type="xsd:hexBinary"/>
    <xsd:element name="ded-set-request" type="xsd:hexBinary"/>
    <xsd:element name="ded-event-notification-request" type="xsd:hexBinary"/>
    <xsd:element name="ded-actionRequest" type="xsd:hexBinary"/>
    <xsd:element name="ded-get-response" type="xsd:hexBinary"/>
    <xsd:element name="ded-set-response" type="xsd:hexBinary"/>
    <xsd:element name="ded-action-response" type="xsd:hexBinary"/>
    <xsd:element name="exception-response" type="ExceptionResponse"/>
    <xsd:element name="access-request" type="Access-Request"/>
    <xsd:element name="access-response" type="Access-Response"/>
    <xsd:element name="general-glo-ciphering" type="General-Glo-Ciphering"/>
    <xsd:element name="general-ded-ciphering" type="General-Ded-Ciphering"/>
    <xsd:element name="general-ciphering" type="General-Ciphering"/>
    <xsd:element name="general-signing" type="General-Signing"/>
    <xsd:element name="general-block-transfer" type="General-Block-Transfer"/>
</xsd:choice>
</xsd:complexType>

<xsd:complexType name="Data-Notification-Confirm">
    <xsd:sequence>
        <xsd:element name="long-invoke-id-and-priority" type="Long-Invoke-Id-And-Priority"/>
        <xsd:element name="date-time" type="xsd:hexBinary"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:simpleType name="Application-context-name">
    <xsd:restriction base="ObjectIdentifier"/>
</xsd:simpleType>

<xsd:simpleType name="AP-title">
    <xsd:restriction base="xsd:hexBinary"/>
</xsd:simpleType>

<xsd:simpleType name="AE-qualifier">
    <xsd:restriction base="xsd:hexBinary"/>
</xsd:simpleType>

<xsd:simpleType name="AP-invocation-identifier">
    <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>

<xsd:simpleType name="AE-invocation-identifier">
    <xsd:restriction base="xsd:integer"/>
</xsd:simpleType>

<xsd:simpleType name="ACSE-requirements">
    <xsd:union memberTypes="BitString">
        <xsd:simpleType>
            <xsd:list>
                <xsd:simpleType>
                    <xsd:restriction base="xsd:token">
                        <xsd:enumeration value="authentication"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:list>
        </xsd:simpleType>
    </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="Mechanism-name">
    <xsd:restriction base="ObjectIdentifier"/>
</xsd:simpleType>

<xsd:simpleType name="Implementation-data">
    <xsd:restriction base="xsd:string"/>
</xsd:simpleType>

<xsd:simpleType name="Association-information">
    <xsd:restriction base="xsd:hexBinary"/>
</xsd:simpleType>

```

```

<xsd:simpleType name="Association-result">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value="accepted"/>
        <xsd:enumeration value="rejected-permanent"/>
        <xsd:enumeration value="rejected-transient"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer"/>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="Release-request-reason">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value="normal"/>
        <xsd:enumeration value="urgent"/>
        <xsd:enumeration value="user-defined"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer"/>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="Release-response-reason">
  <xsd:union>
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value="normal"/>
        <xsd:enumeration value="not-finished"/>
        <xsd:enumeration value="user-defined"/>
      </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType>
      <xsd:restriction base="xsd:integer"/>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="Integer8">
  <xsd:restriction base="xsd:byte"/>
</xsd:simpleType>

<xsd:simpleType name="Integer16">
  <xsd:restriction base="xsd:short"/>
</xsd:simpleType>

<xsd:simpleType name="Integer32">
  <xsd:restriction base="xsd:int"/>
</xsd:simpleType>

<xsd:simpleType name="Integer64">
  <xsd:restriction base="xsd:long"/>
</xsd:simpleType>

<xsd:simpleType name="Unsigned8">
  <xsd:restriction base="xsd:unsignedByte"/>
</xsd:simpleType>

<xsd:simpleType name="Unsigned16">
  <xsd:restriction base="xsd:unsignedShort"/>
</xsd:simpleType>

<xsd:simpleType name="Unsigned32">
  <xsd:restriction base="xsd:unsignedInt"/>
</xsd:simpleType>

```

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	387/614
-----------------------	------------	-----------------------	---------

```

<xsd:simpleType name="Unsigned64">
  <xsd:restriction base="xsd:unsignedLong"/>
</xsd:simpleType>

<xsd:simpleType name="Conformance">
  <xsd:union memberTypes="BitString">
    <xsd:simpleType>
      <xsd:list>
        <xsd:simpleType>
          <xsd:restriction base="xsd:token">
            <xsd:enumeration value="reserved-zero"/>
            <xsd:enumeration value="general-protection"/>
            <xsd:enumeration value="general-block-transfer"/>
            <xsd:enumeration value="read"/>
            <xsd:enumeration value="write"/>
            <xsd:enumeration value="unconfirmed-write"/>
            <xsd:enumeration value="delta-value-encoding"/>
            <xsd:enumeration value="reserved-seven"/>
            <xsd:enumeration value="attribute0-supported-with-set"/>
            <xsd:enumeration value="priority-mgmt-supported"/>
            <xsd:enumeration value="attribute0-supported-with-get"/>
            <xsd:enumeration value="block-transfer-with-get-or-read"/>
            <xsd:enumeration value="block-transfer-with-set-or-write"/>
            <xsd:enumeration value="block-transfer-with-action"/>
            <xsd:enumeration value="multiple-references"/>
            <xsd:enumeration value="information-report"/>
            <xsd:enumeration value="data-notification"/>
            <xsd:enumeration value="access"/>
            <xsd:enumeration value="parameterized-access"/>
            <xsd:enumeration value="get"/>
            <xsd:enumeration value="set"/>
            <xsd:enumeration value="selective-access"/>
            <xsd:enumeration value="event-notification"/>
            <xsd:enumeration value="action"/>
          </xsd:restriction>
        </xsd:simpleType>
      </xsd:list>
    </xsd:simpleType>
  </xsd:union>
</xsd:simpleType>

<xsd:simpleType name="ObjectName">
  <xsd:restriction base="Integer16"/>
</xsd:simpleType>

<xsd:simpleType name="Data-Access-Result">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="success"/>
    <xsd:enumeration value="hardware-fault"/>
    <xsd:enumeration value="temporary-failure"/>
    <xsd:enumeration value="read-write-denied"/>
    <xsd:enumeration value="object-undefined"/>
    <xsd:enumeration value="object-class-inconsistent"/>
    <xsd:enumeration value="object-unavailable"/>
    <xsd:enumeration value="type-unmatched"/>
    <xsd:enumeration value="scope-of-access-violated"/>
    <xsd:enumeration value="data-block-unavailable"/>
    <xsd:enumeration value="long-get-aborted"/>
    <xsd:enumeration value="no-long-get-in-progress"/>
    <xsd:enumeration value="long-set-aborted"/>
    <xsd:enumeration value="no-long-set-in-progress"/>
    <xsd:enumeration value="data-block-number-invalid"/>
    <xsd:enumeration value="other-reason"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Action-Result">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="success"/>
    <xsd:enumeration value="hardware-fault"/>
    <xsd:enumeration value="temporary-failure"/>
    <xsd:enumeration value="read-write-denied"/>
    <xsd:enumeration value="object-undefined"/>
    <xsd:enumeration value="object-class-inconsistent"/>
  </xsd:restriction>
</xsd:simpleType>

```

```

<xsd:enumeration value="object-unavailable"/>
<xsd:enumeration value="type-unmatched"/>
<xsd:enumeration value="scope-of-access-violated"/>
<xsd:enumeration value="data-block-unavailable"/>
<xsd:enumeration value="long-action-aborted"/>
<xsd:enumeration value="no-long-action-in-progress"/>
<xsd:enumeration value="other-reason"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Invoke-Id-And-Priority">
  <xsd:restriction base="Unsigned8"/>
</xsd:simpleType>

<xsd:simpleType name="Long-Invoke-Id-And-Priority">
  <xsd:restriction base="Unsigned32"/>
</xsd:simpleType>

<xsd:simpleType name="Cosem-Class-Id">
  <xsd:restriction base="Unsigned16"/>
</xsd:simpleType>

<xsd:simpleType name="Cosem-Object-Instance-Id">
  <xsd:restriction base="xsd:hexBinary">
    <xsd:length value="6"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Cosem-Object-Attribute-Id">
  <xsd:restriction base="Integer8"/>
</xsd:simpleType>

<xsd:simpleType name="Cosem-Object-Method-Id">
  <xsd:restriction base="Integer8"/>
</xsd:simpleType>

<xsd:simpleType name="Key-Id">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="global-unicast-encryption-key"/>
    <xsd:enumeration value="global-broadcast-encryption-key"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Kek-Id">
  <xsd:restriction base="xsd:token">
    <xsd:enumeration value="master-key"/>
  </xsd:restriction>
</xsd:simpleType>

<xsd:simpleType name="Block-Control">
  <xsd:restriction base="Unsigned8"/>
</xsd:simpleType>

<xsd:complexType name="Authentication-value">
  <xsd:choice>
    <xsd:element name="charstring" type="xsd:string"/>
    <xsd:element name="bitstring" type="BitString"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="AARQ-apdu">
  <xsd:sequence>
    <xsd:element name="protocol-version" minOccurs="0">
      <xsd:simpleType>
        <xsd:union memberTypes="BitString">
          <xsd:simpleType>
            <xsd:list>
              <xsd:simpleType>
                <xsd:restriction base="xsd:token">
                  <xsd:enumeration value="version1"/>
                </xsd:restriction>
              </xsd:simpleType>
            </xsd:list>
          </xsd:simpleType>
        </xsd:union>
      </xsd:simpleType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

```

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	389/614
-----------------------	------------	-----------------------	---------

```

        </xsd:union>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="application-context-name" type="Application-context-name"/>
<xsd:element name="called-AP-title" minOccurs="0" type="AP-title"/>
<xsd:element name="called-AE-qualifier" minOccurs="0" type="AE-qualifier"/>
<xsd:element name="called-AP-invocation-id" minOccurs="0" type="AP-invocation-identifier"/>
<xsd:element name="called-AE-invocation-id" minOccurs="0" type="AE-invocation-identifier"/>
<xsd:element name="calling-AP-title" minOccurs="0" type="AP-title"/>
<xsd:element name="calling-AE-qualifier" minOccurs="0" type="AE-qualifier"/>
<xsd:element name="calling-AP-invocation-id" minOccurs="0" type="AP-invocation-identifier"/>
<xsd:element name="calling-AE-invocation-id" minOccurs="0" type="AE-invocation-identifier"/>
<xsd:element name="sender-acse-requirements" minOccurs="0" type="ACSE-requirements"/>
<xsd:element name="mechanism-name" minOccurs="0" type="Mechanism-name"/>
<xsd:element name="calling-authentication-value" minOccurs="0" type="Authentication-value"/>
<xsd:element name="implementation-information" minOccurs="0" type="Implementation-data"/>
<xsd:element name="user-information" minOccurs="0" type="Association-information"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Associate-source-diagnostic">
    <xsd:choice>
        <xsd:element name="acse-service-user">
            <xsd:simpleType>
                <xsd:union>
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:token">
                            <xsd:enumeration value="null"/>
                            <xsd:enumeration value="no-reason-given"/>
                            <xsd:enumeration value="application-context-name-not-supported"/>
                            <xsd:enumeration value="calling-AP-title-not-recognized"/>
                            <xsd:enumeration value="calling-AP-invocation-identifier-not-recognized"/>
                            <xsd:enumeration value="calling-AE-qualifier-not-recognized"/>
                            <xsd:enumeration value="calling-AE-invocation-identifier-not-recognized"/>
                            <xsd:enumeration value="called-AP-title-not-recognized"/>
                            <xsd:enumeration value="called-AP-invocation-identifier-not-recognized"/>
                            <xsd:enumeration value="called-AE-qualifier-not-recognized"/>
                            <xsd:enumeration value="called-AE-invocation-identifier-not-recognized"/>
                            <xsd:enumeration value="authentication-mechanism-name-not-recognised"/>
                            <xsd:enumeration value="authentication-mechanism-name-required"/>
                            <xsd:enumeration value="authentication-failure"/>
                            <xsd:enumeration value="authentication-required"/>
                        </xsd:restriction>
                    </xsd:simpleType>
                    <xsd:simpleType>
                        <xsd:restriction base="xsd:integer"/>
                    </xsd:simpleType>
                </xsd:union>
            </xsd:simpleType>
        </xsd:element>
    </xsd:choice>
</xsd:complexType>

<xsd:element name="acse-service-provider">
    <xsd:simpleType>
        <xsd:union>
            <xsd:simpleType>
                <xsd:restriction base="xsd:token">
                    <xsd:enumeration value="null"/>
                    <xsd:enumeration value="no-reason-given"/>
                    <xsd:enumeration value="no-common-acse-version"/>
                </xsd:restriction>
            </xsd:simpleType>
            <xsd:simpleType>
                <xsd:restriction base="xsd:integer"/>
            </xsd:simpleType>
        </xsd:union>
    </xsd:simpleType>
</xsd:element>
</xsd:choice>
</xsd:complexType>

<xsd:complexType name="AARE-apdu">
    <xsd:sequence>
        <xsd:element name="protocol-version" minOccurs="0">
            <xsd:simpleType>
                <xsd:union memberTypes="BitString">

```

```

<xsd:simpleType>
  <xsd:list>
    <xsd:simpleType>
      <xsd:restriction base="xsd:token">
        <xsd:enumeration value="version1"/>
      </xsd:restriction>
    </xsd:simpleType>
  </xsd:list>
</xsd:simpleType>
</xsd:union>
</xsd:simpleType>
</xsd:element>
<xsd:element name="application-context-name" type="Application-context-name"/>
<xsd:element name="result" type="Association-result"/>
<xsd:element name="result-source-diagnostic" type="Associate-source-diagnostic"/>
<xsd:element name="responding-AP-title" minOccurs="0" type="AP-title"/>
<xsd:element name="responding-AE-qualifier" minOccurs="0" type="AE-qualifier"/>
<xsd:element name="responding-AP-invocation-id" minOccurs="0" type="AP-invocation-identifier"/>
<xsd:element name="responding-AE-invocation-id" minOccurs="0" type="AE-invocation-identifier"/>
<xsd:element name="responder-acse-requirements" minOccurs="0" type="ACSE-requirements"/>
<xsd:element name="mechanism-name" minOccurs="0" type="Mechanism-name"/>
<xsd:element name="responding-authentication-value" minOccurs="0" type="Authentication-value"/>
<xsd:element name="implementation-information" minOccurs="0" type="Implementation-data"/>
<xsd:element name="user-information" minOccurs="0" type="Association-information"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="RLRQ-apdu">
  <xsd:sequence>
    <xsd:element name="reason" minOccurs="0" type="Release-request-reason"/>
    <xsd:element name="user-information" minOccurs="0" type="Association-information"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="RLRE-apdu">
  <xsd:sequence>
    <xsd:element name="reason" minOccurs="0" type="Release-response-reason"/>
    <xsd:element name="user-information" minOccurs="0" type="Association-information"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="InitiateRequest">
  <xsd:sequence>
    <xsd:element name="dedicated-key" minOccurs="0" type="xsd:hexBinary"/>
    <xsd:element name="response-allowed" default="true" type="xsd:boolean"/>
    <xsd:element name="proposed-quality-of-service" minOccurs="0" type="Integer8"/>
    <xsd:element name="proposed-dlms-version-number" type="Unsigned8"/>
    <xsd:element name="proposed-conformance" type="Conformance"/>
    <xsd:element name="client-max-receive-pdu-size" type="Unsigned16"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="TypeDescription">
  <xsd:choice>
    <xsd:element name="null-data" type="NULL"/>
    <xsd:element name="array">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="number-of-elements" type="Unsigned16"/>
          <xsd:element name="type-description" type="TypeDescription"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="structure">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="TypeDescription" type="TypeDescription"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="boolean" type="NULL"/>
    <xsd:element name="bit-string" type="NULL"/>
    <xsd:element name="double-long" type="NULL"/>
    <xsd:element name="double-long-unsigned" type="NULL"/>
  </xsd:choice>
</xsd:complexType>

```

```

<xsd:element name="octet-string" type="NULL"/>
<xsd:element name="visible-string" type="NULL"/>
<xsd:element name="utf8-string" type="NULL"/>
<xsd:element name="bcd" type="NULL"/>
<xsd:element name="integer" type="NULL"/>
<xsd:element name="long" type="NULL"/>
<xsd:element name="unsigned" type="NULL"/>
<xsd:element name="long-unsigned" type="NULL"/>
<xsd:element name="long64" type="NULL"/>
<xsd:element name="long64-unsigned" type="NULL"/>
<xsd:element name="enum" type="NULL"/>
<xsd:element name="float32" type="NULL"/>
<xsd:element name="float64" type="NULL"/>
<xsd:element name="date-time" type="NULL"/>
<xsd:element name="date" type="NULL"/>
<xsd:element name="time" type="NULL"/>
<xsd:element name="dont-care" type="NULL"/>
</xsd:choice>
</xsd:complexType>

<xsd:complexType name="SequenceOfData">
    <xsd:choice minOccurs="0" maxOccurs="unbounded">
        <xsd:element name="null-data" type="NULL"/>
        <xsd:element name="array" type="SequenceOfData"/>
        <xsd:element name="structure" type="SequenceOfData"/>
        <xsd:element name="boolean" type="xsd:boolean"/>
        <xsd:element name="bit-string" type="BitString"/>
        <xsd:element name="double-long" type="Integer32"/>
        <xsd:element name="double-long-unsigned" type="Unsigned32"/>
        <xsd:element name="octet-string" type="xsd:hexBinary"/>
        <xsd:element name="visible-string" type="xsd:string"/>
        <xsd:element name="utf8-string" type="xsd:string"/>
        <xsd:element name="bcd" type="Integer8"/>
        <xsd:element name="integer" type="Integer8"/>
        <xsd:element name="long" type="Integer16"/>
        <xsd:element name="unsigned" type="Unsigned8"/>
        <xsd:element name="long-unsigned" type="Unsigned16"/>
        <xsd:element name="compact-array">
            <xsd:complexType>
                <xsd:sequence>
                    <xsd:element name="contents-description" type="TypeDescription"/>
                    <xsd:element name="array-contents" type="xsd:hexBinary"/>
                </xsd:sequence>
            </xsd:complexType>
        </xsd:element>
        <xsd:element name="long64" type="Integer64"/>
        <xsd:element name="long64-unsigned" type="Unsigned64"/>
        <xsd:element name="enum" type="Unsigned8"/>
        <xsd:element name="float32" type="xsd:float"/>
        <xsd:element name="float64" type="xsd:double"/>
        <xsd:element name="date-time">
            <xsd:simpleType>
                <xsd:restriction base="xsd:hexBinary">
                    <xsd:length value="12"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="date">
            <xsd:simpleType>
                <xsd:restriction base="xsd:hexBinary">
                    <xsd:length value="5"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="time">
            <xsd:simpleType>
                <xsd:restriction base="xsd:hexBinary">
                    <xsd:length value="4"/>
                </xsd:restriction>
            </xsd:simpleType>
        </xsd:element>
        <xsd:element name="delta-integer" type="Integer8"/>
        <xsd:element name="delta-long" type="Integer16"/>
        <xsd:element name="delta-double-long" type="Integer32"/>
    </xsd:choice>
</xsd:complexType>

```

```

<xsd:element name="delta-unsigned" type="Unsigned8"/>
<xsd:element name="delta-long-unsigned" type="Unsigned16"/>
<xsd:element name="delta-double-long-unsigned" type="Unsigned32"/>
<xsd:element name="dont-care" type="NULL"/>
</xsd:choice>
</xsd:complexType>

<xsd:complexType name="Data">
  <xsd:choice>
    <xsd:element name="null-data" type="NULL"/>
    <xsd:element name="array" type="SequenceOfData"/>
    <xsd:element name="structure" type="SequenceOfData"/>
    <xsd:element name="boolean" type="xsd:boolean"/>
    <xsd:element name="bit-string" type="BitString"/>
    <xsd:element name="double-long" type="Integer32"/>
    <xsd:element name="double-long-unsigned" type="Unsigned32"/>
    <xsd:element name="octet-string" type="xsd:hexBinary"/>
    <xsd:element name="visible-string" type="xsd:string"/>
    <xsd:element name="utf8-string" type="xsd:string"/>
    <xsd:element name="bcd" type="Integer8"/>
    <xsd:element name="integer" type="Integer8"/>
    <xsd:element name="long" type="Integer16"/>
    <xsd:element name="unsigned" type="Unsigned8"/>
    <xsd:element name="long-unsigned" type="Unsigned16"/>
    <xsd:element name="compact-array">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="contents-description" type="TypeDescription"/>
          <xsd:element name="array-contents" type="xsd:hexBinary"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="long64" type="Integer64"/>
    <xsd:element name="long64-unsigned" type="Unsigned64"/>
    <xsd:element name="enum" type="Unsigned8"/>
    <xsd:element name="float32" type="xsd:float"/>
    <xsd:element name="float64" type="xsd:double"/>
    <xsd:element name="date-time">
      <xsd:simpleType>
        <xsd:restriction base="xsd:hexBinary">
          <xsd:length value="12"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="date">
      <xsd:simpleType>
        <xsd:restriction base="xsd:hexBinary">
          <xsd:length value="5"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="time">
      <xsd:simpleType>
        <xsd:restriction base="xsd:hexBinary">
          <xsd:length value="4"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="delta-integer" type="Integer8"/>
    <xsd:element name="delta-long" type="Integer16"/>
    <xsd:element name="delta-double-long" type="Integer32"/>
    <xsd:element name="delta-unsigned" type="Unsigned8"/>
    <xsd:element name="delta-long-unsigned" type="Unsigned16"/>
    <xsd:element name="delta-double-long-unsigned" type="Unsigned32"/>
    <xsd:element name="dont-care" type="NULL"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="Parameterized-Access">
  <xsd:sequence>
    <xsd:element name="variable-name" type="ObjectName"/>
    <xsd:element name="selector" type="Unsigned8"/>
    <xsd:element name="parameter" type="Data"/>
  </xsd:sequence>
</xsd:complexType>

```

```

</xsd:complexType>

<xsd:complexType name="Block-Number-Access">
  <xsd:sequence>
    <xsd:element name="block-number" type="Unsigned16"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Read-Data-Block-Access">
  <xsd:sequence>
    <xsd:element name="last-block" type="xsd:boolean"/>
    <xsd:element name="block-number" type="Unsigned16"/>
    <xsd:element name="raw-data" type="xsd:hexBinary"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Write-Data-Block-Access">
  <xsd:sequence>
    <xsd:element name="last-block" type="xsd:boolean"/>
    <xsd:element name="block-number" type="Unsigned16"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Variable-Access-Specification">
  <xsd:choice>
    <xsd:element name="variable-name" type="ObjectName"/>
    <xsd:element name="parameterized-access" type="Parameterized-Access"/>
    <xsd:element name="block-number-access" type="Block-Number-Access"/>
    <xsd:element name="read-data-block-access" type="Read-Data-Block-Access"/>
    <xsd:element name="write-data-block-access" type="Write-Data-Block-Access"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="ReadRequest">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="Variable-Access-Specification" type="Variable-Access-Specification"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="WriteRequest">
  <xsd:sequence>
    <xsd:element name="variable-access-specification">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Variable-Access-Specification" type="Variable-Access-Specification"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="list-of-data">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Data" type="Data"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="InitiateResponse">
  <xsd:sequence>
    <xsd:element name="negotiated-quality-of-service" minOccurs="0" type="Integer8"/>
    <xsd:element name="negotiated-dlms-version-number" type="Unsigned8"/>
    <xsd:element name="negotiated-conformance" type="Conformance"/>
    <xsd:element name="server-max-receive-pdu-size" type="Unsigned16"/>
    <xsd:element name="vaa-name" type="ObjectName"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Data-Block-Result">
  <xsd:sequence>
    <xsd:element name="last-block" type="xsd:boolean"/>
    <xsd:element name="block-number" type="Unsigned16"/>
    <xsd:element name="raw-data" type="xsd:hexBinary"/>
  </xsd:sequence>

```

```

</xsd:complexType>

<xsd:complexType name="ReadResponse">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="CHOICE">
      <xsd:complexType>
        <xsd:choice>
          <xsd:element name="data" type="Data"/>
          <xsd:element name="data-access-error" type="Data-Access-Result"/>
          <xsd:element name="data-block-result" type="Data-Block-Result"/>
          <xsd:element name="block-number" type="Unsigned16"/>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="WriteResponse">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="CHOICE">
      <xsd:complexType>
        <xsd:choice>
          <xsd:element name="success" type="NULL"/>
          <xsd:element name="data-access-error" type="Data-Access-Result"/>
          <xsd:element name="block-number" type="Unsigned16"/>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ServiceError">
  <xsd:choice>
    <xsd:element name="application-reference">
      <xsd:simpleType>
        <xsd:restriction base="xsd:token">
          <xsd:enumeration value="other"/>
          <xsd:enumeration value="time-elapsed"/>
          <xsd:enumeration value="application-unreachable"/>
          <xsd:enumeration value="application-reference-invalid"/>
          <xsd:enumeration value="application-context-unsupported"/>
          <xsd:enumeration value="provider-communication-error"/>
          <xsd:enumeration value="deciphering-error"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="hardware-resource">
      <xsd:simpleType>
        <xsd:restriction base="xsd:token">
          <xsd:enumeration value="other"/>
          <xsd:enumeration value="memory-unavailable"/>
          <xsd:enumeration value="processor-resource-unavailable"/>
          <xsd:enumeration value="mass-storage-unavailable"/>
          <xsd:enumeration value="other-resource-unavailable"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="vde-state-error">
      <xsd:simpleType>
        <xsd:restriction base="xsd:token">
          <xsd:enumeration value="other"/>
          <xsd:enumeration value="no-dlms-context"/>
          <xsd:enumeration value="loading-data-set"/>
          <xsd:enumeration value="status-nochange"/>
          <xsd:enumeration value="status-inoperable"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="service">
      <xsd:simpleType>
        <xsd:restriction base="xsd:token">
          <xsd:enumeration value="other"/>
          <xsd:enumeration value="pdu-size"/>
          <xsd:enumeration value="service-unsupported"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>

```

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	395/614
-----------------------	------------	-----------------------	---------

```

        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="definition">
    <xsd:simpleType>
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="other"/>
            <xsd:enumeration value="object-undefined"/>
            <xsd:enumeration value="object-class-inconsistent"/>
            <xsd:enumeration value="object-attribute-inconsistent"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="access">
    <xsd:simpleType>
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="other"/>
            <xsd:enumeration value="scope-of-access-violated"/>
            <xsd:enumeration value="object-access-violated"/>
            <xsd:enumeration value="hardware-fault"/>
            <xsd:enumeration value="object-unavailable"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="initiate">
    <xsd:simpleType>
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="other"/>
            <xsd:enumeration value="dlms-version-too-low"/>
            <xsd:enumeration value="incompatible-conformance"/>
            <xsd:enumeration value="pdu-size-too-short"/>
            <xsd:enumeration value="refused-by-the-VDE-Handler"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="load-data-set">
    <xsd:simpleType>
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="other"/>
            <xsd:enumeration value="primitive-out-of-sequence"/>
            <xsd:enumeration value="not-loadable"/>
            <xsd:enumeration value="dataset-size-too-large"/>
            <xsd:enumeration value="not-awaited-segment"/>
            <xsd:enumeration value="interpretation-failure"/>
            <xsd:enumeration value="storage-failure"/>
            <xsd:enumeration value="data-set-not-ready"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
<xsd:element name="task">
    <xsd:simpleType>
        <xsd:restriction base="xsd:token">
            <xsd:enumeration value="other"/>
            <xsd:enumeration value="no-remote-control"/>
            <xsd:enumeration value="ti-stopped"/>
            <xsd:enumeration value="ti-running"/>
            <xsd:enumeration value="ti-unusable"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:element>
</xsd:choice>
</xsd:complexType>

<xsd:complexType name="ConfirmedServiceError">
    <xsd:choice>
        <xsd:element name="initiateError" type="ServiceError"/>
        <xsd:element name="getStatus" type="ServiceError"/>
        <xsd:element name="getNameList" type="ServiceError"/>
        <xsd:element name="getVariableAttribute" type="ServiceError"/>
        <xsd:element name="read" type="ServiceError"/>
        <xsd:element name="write" type="ServiceError"/>
        <xsd:element name="getDataSetAttribute" type="ServiceError"/>
        <xsd:element name="getTIAttribute" type="ServiceError"/>
        <xsd:element name="changeScope" type="ServiceError"/>
    </xsd:choice>

```

```

<xsd:element name="start" type="ServiceError"/>
<xsd:element name="stop" type="ServiceError"/>
<xsd:element name="resume" type="ServiceError"/>
<xsd:element name="makeUsable" type="ServiceError"/>
<xsd:element name="initiateLoad" type="ServiceError"/>
<xsd:element name="loadSegment" type="ServiceError"/>
<xsd:element name="terminateLoad" type="ServiceError"/>
<xsd:element name="initiateUpLoad" type="ServiceError"/>
<xsd:element name="upLoadSegment" type="ServiceError"/>
<xsd:element name="terminateUpLoad" type="ServiceError"/>
</xsd:choice>
</xsd:complexType>

<xsd:complexType name="Notification-Body">
  <xsd:sequence>
    <xsd:element name="data-value" type="Data"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Data-Notification">
  <xsd:sequence>
    <xsd:element name="long-invoke-id-and-priority" type="Long-Invoke-Id-And-Priority"/>
    <xsd:element name="date-time" type="xsd:hexBinary"/>
    <xsd:element name="notification-body" type="Notification-Body"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="UnconfirmedWriteRequest">
  <xsd:sequence>
    <xsd:element name="variable-access-specification">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Variable-Access-Specification" type="Variable-Access-Specification"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="list-of-data">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Data" type="Data"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="InformationReportRequest">
  <xsd:sequence>
    <xsd:element name="current-time" minOccurs="0" type="xsd:dateTime"/>
    <xsd:element name="variable-access-specification">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Variable-Access-Specification" type="Variable-Access-Specification"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="list-of-data">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Data" type="Data"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Cosem-Attribute-Descriptor">
  <xsd:sequence>
    <xsd:element name="class-id" type="Cosem-Class-Id"/>
    <xsd:element name="instance-id" type="Cosem-Object-Instance-Id"/>
    <xsd:element name="attribute-id" type="Cosem-Object-Attribute-Id"/>
  </xsd:sequence>
</xsd:complexType>

```

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	397/614
-----------------------	------------	-----------------------	---------

```

<xsd:complexType name="Selective-Access-Descriptor">
  <xsd:sequence>
    <xsd:element name="access-selector" type="Unsigned8"/>
    <xsd:element name="access-parameters" type="Data"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Get-Request-Normal">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
    <xsd:element name="access-selection" minOccurs="0" type="Selective-Access-Descriptor"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Get-Request-Next">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="block-number" type="Unsigned32"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Cosem-Attribute-Descriptor-With-Selection">
  <xsd:sequence>
    <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
    <xsd:element name="access-selection" minOccurs="0" type="Selective-Access-Descriptor"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Get-Request-With-List">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="attribute-descriptor-list">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Cosem-Attribute-Descriptor-With-Selection" type="Cosem-Attribute-Descriptor-With-Selection"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Get-Request">
  <xsd:choice>
    <xsd:element name="get-request-normal" type="Get-Request-Normal"/>
    <xsd:element name="get-request-next" type="Get-Request-Next"/>
    <xsd:element name="get-request-with-list" type="Get-Request-With-List"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="Set-Request-Normal">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
    <xsd:element name="access-selection" minOccurs="0" type="Selective-Access-Descriptor"/>
    <xsd:element name="value" type="Data"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="DataBlock-SA">
  <xsd:sequence>
    <xsd:element name="last-block" type="xsd:boolean"/>
    <xsd:element name="block-number" type="Unsigned32"/>
    <xsd:element name="raw-data" type="xsd:hexBinary"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Request-With-First-Datablock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
    <xsd:element name="access-selection" minOccurs="0" type="Selective-Access-Descriptor"/>
    <xsd:element name="datablock" type="DataBlock-SA"/>
  </xsd:sequence>
</xsd:complexType>

```

```

</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Request-With-Datablock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="datablock" type="DataBlock-SA"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Request-With-List">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="attribute-descriptor-list">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Cosem-Attribute-Descriptor-With-Selection" type="Cosem-Attribute-Descriptor-With-Selection"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="value-list">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Data" type="Data"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Request-With-List-And-First-Datablock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="attribute-descriptor-list">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Cosem-Attribute-Descriptor-With-Selection" type="Cosem-Attribute-Descriptor-With-Selection"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="datablock" type="DataBlock-SA"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Request">
  <xsd:choice>
    <xsd:element name="set-request-normal" type="Set-Request-Normal"/>
    <xsd:element name="set-request-with-first-datablock" type="Set-Request-With-First-Datablock"/>
    <xsd:element name="set-request-with-datablock" type="Set-Request-With-Datablock"/>
    <xsd:element name="set-request-with-list" type="Set-Request-With-List"/>
    <xsd:element name="set-request-with-list-and-first-datablock" type="Set-Request-With-List-And-First-Datablock"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="EventNotificationRequest">
  <xsd:sequence>
    <xsd:element name="time" minOccurs="0" type="xsd:hexBinary"/>
    <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
    <xsd:element name="attribute-value" type="Data"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Cosem-Method-Descriptor">
  <xsd:sequence>
    <xsd:element name="class-id" type="Cosem-Class-Id"/>
    <xsd:element name="instance-id" type="Cosem-Object-Instance-Id"/>
    <xsd:element name="method-id" type="Cosem-Object-Method-Id"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Request-Normal">

```

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	399/614
-----------------------	------------	-----------------------	---------

```

<xsd:sequence>
  <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
  <xsd:element name="cosem-method-descriptor" type="Cosem-Method-Descriptor"/>
  <xsd:element name="method-invocation-parameters" minOccurs="0" type="Data"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Request-Next-Pblock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="block-number" type="Unsigned32"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Request-With-List">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="cosem-method-descriptor-list">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Cosem-Method-Descriptor" type="Cosem-Method-Descriptor"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="method-invocation-parameters">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Data" type="Data"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Request-With-First-Pblock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="cosem-method-descriptor" type="Cosem-Method-Descriptor"/>
    <xsd:element name="pblock" type="DataBlock-SA"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Request-With-List-And-First-Pblock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="cosem-method-descriptor-list">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Cosem-Method-Descriptor" type="Cosem-Method-Descriptor"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
    <xsd:element name="pblock" type="DataBlock-SA"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Request-With-Pblock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="pblock" type="DataBlock-SA"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Request">
  <xsd:choice>
    <xsd:element name="action-request-normal" type="Action-Request-Normal"/>
    <xsd:element name="action-request-next-pblock" type="Action-Request-Next-Pblock"/>
    <xsd:element name="action-request-with-list" type="Action-Request-With-List"/>
    <xsd:element name="action-request-with-first-pblock" type="Action-Request-With-First-Pblock"/>
    <xsd:element name="action-request-with-list-and-first-pblock" type="Action-Request-With-List-And-First-Pblock"/>
    <xsd:element name="action-request-with-pblock" type="Action-Request-With-Pblock"/>
  </xsd:choice>
</xsd:complexType>

```

```

<xsd:complexType name="Get-Data-Result">
  <xsd:choice>
    <xsd:element name="data" type="Data"/>
    <xsd:element name="data-access-result" type="Data-Access-Result"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="Get-Response-Normal">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="result" type="Get-Data-Result"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="DataBlock-G">
  <xsd:sequence>
    <xsd:element name="last-block" type="xsd:boolean"/>
    <xsd:element name="block-number" type="Unsigned32"/>
    <xsd:element name="result">
      <xsd:complexType>
        <xsd:choice>
          <xsd:element name="raw-data" type="xsd:hexBinary"/>
          <xsd:element name="data-access-result" type="Data-Access-Result"/>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Get-Response-With-Datablock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="result" type="DataBlock-G"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Get-Response-With-List">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="result">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Get-Data-Result" type="Get-Data-Result"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Get-Response">
  <xsd:choice>
    <xsd:element name="get-response-normal" type="Get-Response-Normal"/>
    <xsd:element name="get-response-with-datablock" type="Get-Response-With-Datablock"/>
    <xsd:element name="get-response-with-list" type="Get-Response-With-List"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="Set-Response-Normal">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="result" type="Data-Access-Result"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Response-Datablock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="block-number" type="Unsigned32"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Response-Last-Datablock">
  <xsd:sequence>

```

```

<xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
<xsd:element name="result" type="Data-Access-Result"/>
<xsd:element name="block-number" type="Unsigned32"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Response-Last-Datablock-With-List">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="result">
      <xsd:simpleType>
        <xsd:list itemType="Data-Access-Result"/>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="block-number" type="Unsigned32"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Response-With-List">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="result">
      <xsd:simpleType>
        <xsd:list itemType="Data-Access-Result"/>
      </xsd:simpleType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Set-Response">
  <xsd:choice>
    <xsd:element name="set-response-normal" type="Set-Response-Normal"/>
    <xsd:element name="set-response-datablock" type="Set-Response-Datablock"/>
    <xsd:element name="set-response-last-datablock" type="Set-Response-Last-Datablock"/>
    <xsd:element name="set-response-last-datablock-with-list" type="Set-Response-Last-Datablock-With-
List"/>
    <xsd:element name="set-response-with-list" type="Set-Response-With-List"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="Action-Response-With-Optional-Data">
  <xsd:sequence>
    <xsd:element name="result" type="Action-Result"/>
    <xsd:element name="return-parameters" minOccurs="0" type="Get-Data-Result"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Response-Normal">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="single-response" type="Action-Response-With-Optional-Data"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Response-With-Pblock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="pblock" type="DataBlock-SA"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Response-With-List">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="list-of-responses">
      <xsd:complexType>
        <xsd:sequence minOccurs="0" maxOccurs="unbounded">
          <xsd:element name="Action-Response-With-Optional-Data" type="Action-Response-With-Optional-
Data"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

```

```

<xsd:complexType name="Action-Response-Next-Pblock">
  <xsd:sequence>
    <xsd:element name="invoke-id-and-priority" type="Invoke-Id-And-Priority"/>
    <xsd:element name="block-number" type="Unsigned32"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Action-Response">
  <xsd:choice>
    <xsd:element name="action-response-normal" type="Action-Response-Normal"/>
    <xsd:element name="action-response-with-pblock" type="Action-Response-With-Pblock"/>
    <xsd:element name="action-response-with-list" type="Action-Response-With-List"/>
    <xsd:element name="action-response-next-pblock" type="Action-Response-Next-Pblock"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="ExceptionResponse">
  <xsd:sequence>
    <xsd:element name="state-error">
      <xsd:simpleType>
        <xsd:restriction base="xsd:token">
          <xsd:enumeration value="service-not-allowed"/>
          <xsd:enumeration value="service-unknown"/>
        </xsd:restriction>
      </xsd:simpleType>
    </xsd:element>
    <xsd:element name="service-error">
      <xsd:complexType>
        <xsd:choice>
          <xsd:element name="operation-not-possible" type="NULL"/>
          <xsd:element name="service-not-supported" type="NULL"/>
          <xsd:element name="other-reason" type="NULL"/>
          <xsd:element name="pdu-too-long" type="NULL"/>
          <xsd:element name="deciphering-error" type="NULL"/>
          <xsd:element name="invocation-counter-error" type="Unsigned32"/>
        </xsd:choice>
      </xsd:complexType>
    </xsd:element>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Get">
  <xsd:sequence>
    <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Set">
  <xsd:sequence>
    <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Action">
  <xsd:sequence>
    <xsd:element name="cosem-method-descriptor" type="Cosem-Method-Descriptor"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Get-With-Selection">
  <xsd:sequence>
    <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
    <xsd:element name="access-selection" type="Selective-Access-Descriptor"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Set-With-Selection">
  <xsd:sequence>
    <xsd:element name="cosem-attribute-descriptor" type="Cosem-Attribute-Descriptor"/>
    <xsd:element name="access-selection" type="Selective-Access-Descriptor"/>
  </xsd:sequence>
</xsd:complexType>

```

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	403/614
-----------------------	------------	-----------------------	---------

```

<xsd:complexType name="Access-Request-Specification">
  <xsd:choice>
    <xsd:element name="access-request-get" type="Access-Request-Get"/>
    <xsd:element name="access-request-set" type="Access-Request-Set"/>
    <xsd:element name="access-request-action" type="Access-Request-Action"/>
    <xsd:element name="access-request-get-with-selection" type="Access-Request-Get-With-Selection"/>
    <xsd:element name="access-request-set-with-selection" type="Access-Request-Set-With-Selection"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="List-Of-Access-Request-Specification">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="Access-Request-Specification" type="Access-Request-Specification"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="List-Of-Data">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="Data" type="Data"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request-Body">
  <xsd:sequence>
    <xsd:element name="access-request-specification" type="List-Of-Access-Request-Specification"/>
    <xsd:element name="access-request-list-of-data" type="List-Of-Data"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Request">
  <xsd:sequence>
    <xsd:element name="long-invoke-id-and-priority" type="Long-Invoke-Id-And-Priority"/>
    <xsd:element name="date-time" type="xsd:hexBinary"/>
    <xsd:element name="access-request-body" type="Access-Request-Body"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Response-Get">
  <xsd:sequence>
    <xsd:element name="result" type="Data-Access-Result"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Response-Set">
  <xsd:sequence>
    <xsd:element name="result" type="Data-Access-Result"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Response-Action">
  <xsd:sequence>
    <xsd:element name="result" type="Action-Result"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Response-Specification">
  <xsd:choice>
    <xsd:element name="access-response-get" type="Access-Response-Get"/>
    <xsd:element name="access-response-set" type="Access-Response-Set"/>
    <xsd:element name="access-response-action" type="Access-Response-Action"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="List-Of-Access-Response-Specification">
  <xsd:sequence minOccurs="0" maxOccurs="unbounded">
    <xsd:element name="Access-Response-Specification" type="Access-Response-Specification"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Response-Body">
  <xsd:sequence>
    <xsd:element name="access-request-specification" minOccurs="0" type="List-Of-Access-Request-Specification"/>
    <xsd:element name="access-response-list-of-data" type="List-Of-Data"/>
  </xsd:sequence>
</xsd:complexType>

```

```

<xsd:element name="access-response-specification" type="List-Of-Access-Response-Specification"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Access-Response">
  <xsd:sequence>
    <xsd:element name="long-invoke-id-and-priority" type="Long-Invoke-Id-And-Priority"/>
    <xsd:element name="date-time" type="xsd:hexBinary"/>
    <xsd:element name="access-response-body" type="Access-Response-Body"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="General-Glo-Ciphering">
  <xsd:sequence>
    <xsd:element name="system-title" type="xsd:hexBinary"/>
    <xsd:element name="ciphered-content" type="xsd:hexBinary"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="General-Ded-Ciphering">
  <xsd:sequence>
    <xsd:element name="system-title" type="xsd:hexBinary"/>
    <xsd:element name="ciphered-content" type="xsd:hexBinary"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Identified-Key">
  <xsd:sequence>
    <xsd:element name="key-id" type="Key-Id"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Wrapped-Key">
  <xsd:sequence>
    <xsd:element name="kek-id" type="Kek-Id"/>
    <xsd:element name="key-ciphered-data" type="xsd:hexBinary"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Agreed-Key">
  <xsd:sequence>
    <xsd:element name="key-parameters" type="xsd:hexBinary"/>
    <xsd:element name="key-ciphered-data" type="xsd:hexBinary"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="Key-Info">
  <xsd:choice>
    <xsd:element name="identified-key" type="Identified-Key"/>
    <xsd:element name="wrapped-key" type="Wrapped-Key"/>
    <xsd:element name="agreed-key" type="Agreed-Key"/>
  </xsd:choice>
</xsd:complexType>

<xsd:complexType name="General-Ciphering">
  <xsd:sequence>
    <xsd:element name="transaction-id" type="xsd:hexBinary"/>
    <xsd:element name="originator-system-title" type="xsd:hexBinary"/>
    <xsd:element name="recipient-system-title" type="xsd:hexBinary"/>
    <xsd:element name="date-time" type="xsd:hexBinary"/>
    <xsd:element name="other-information" type="xsd:hexBinary"/>
    <xsd:element name="key-info" minOccurs="0" type="Key-Info"/>
    <xsd:element name="ciphered-content" type="xsd:hexBinary"/>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="General-Signing">
  <xsd:sequence>
    <xsd:element name="transaction-id" type="xsd:hexBinary"/>
    <xsd:element name="originator-system-title" type="xsd:hexBinary"/>
    <xsd:element name="recipient-system-title" type="xsd:hexBinary"/>
    <xsd:element name="date-time" type="xsd:hexBinary"/>
    <xsd:element name="other-information" type="xsd:hexBinary"/>
    <xsd:element name="content" type="xsd:hexBinary"/>
  </xsd:sequence>
</xsd:complexType>

```

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	405/614
-----------------------	------------	-----------------------	---------

```
<xsd:element name="signature" type="xsd:hexBinary"/>
</xsd:sequence>
</xsd:complexType>

<xsd:complexType name="General-Block-Transfer">
<xsd:sequence>
<xsd:element name="block-control" type="Block-Control"/>
<xsd:element name="block-number" type="Unsigned16"/>
<xsd:element name="block-number-ack" type="Unsigned16"/>
<xsd:element name="block-data" type="xsd:hexBinary"/>
</xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

## 10 Using the DLMS/COSEM application layer in various communications profiles

### 10.1 Communication profile specific elements

#### 10.1.1 General

The COSEM interface model, specified in DLMS UA 1000-1 has been designed for use with a variety of communication profiles for exchanging data over various communication media. In each such profile, the application layer is the DLMS/COSEM AL, providing the xDLMS services to access attributes and methods of COSEM objects. For each communication profile, the following elements must be specified:

- the targeted communication environments;
- the structure of the profile (the set of protocol layers);
- the identification/addressing scheme;
- mapping of the DLMS/COSEM AL services to the service set provided and used by the supporting protocol layer;
- communication profile specific parameters of the DLMS/COSEM AL services;
- other specific considerations/constraints for using certain services within a given profile.

#### 10.1.2 Targeted communication environments

This part identifies the communication environments, for which the given communication profile is specified.

#### 10.1.3 The structure of the profile

This part specifies the protocol layers included in the given profile.

#### 10.1.4 Identification and addressing schemes

This part describes the identification and addressing schemes specific for the profile.

As described in DLMS UA 1000-1 Part 2 Ed.15:2021, 4.1.7, devices are modelled in COSEM as physical devices, containing one or more logical devices. In the COSEM client/server type model, data exchange takes place within AAs, between a COSEM client AP and a COSEM Logical Device, playing the role of a server AP.

To be able to establish the required AA and then exchanging data with the help of the supporting layer protocols, the client- and server APs must be identified and addressed, according to the rules of a communication profile. At least the following elements need to be identified / addressed:

- physical devices hosting clients and servers;
- client- and server APs;

The client- and server APs also identify the AAs.

#### 10.1.5 Supporting protocol layer services and service mapping

This part specifies the service mapping between the services requested by the DLMS/COSEM AL and the services provided by its supporting protocol layer.

In each communication profile, the DLMS/COSEM AL provides the same set of services to the client- and server APs. However, the supporting protocol layer in the various profiles provides a different set of services to the service user AL.

The service mapping specifies how the AL is using the services of its supporting protocol layer to provide ACSE and xDLMS services to its service user. For this purpose generally MSCs are used showing the sequence of the events following a service invocation by the COSEM AP.

#### 10.1.6 Communication profile specific parameters of the DLMS/COSEM AL services

In DLMS/COSEM, only the COSEM-OPEN service has communication profile specific parameters. Their values and use are defined as part of the communication profile specification.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	407/614
-----------------------	------------	-----------------------	---------

### 10.1.7 Specific considerations / constraints using certain services within a given profile

The availability and the protocol of some of the services may depend on the communication profile. These elements are specified as part of the communication profile specification.

## 10.2 The 3-layer, connection-oriented, HDLC based communication profile

### 10.2.1 Targeted communication environments

The 3-layer, CO, HDLC based profile is suitable for local data exchange with devices equipment via direct connection, or remote data exchange via the PSTN or GSM networks.

### 10.2.2 The structure of the profile

This profile is based on a three-layer (collapsed) OSI protocol architecture:

- the DLMS/COSEM AL, specified in clause 9;
- the data link layer based on the HDLC standard, specified in Clause 8;
- the physical layer; specified in Clause 5. The use of the PhL for the purposes of direct local data exchange using an optical port or a current loop physical interface is specified in Clause 6.

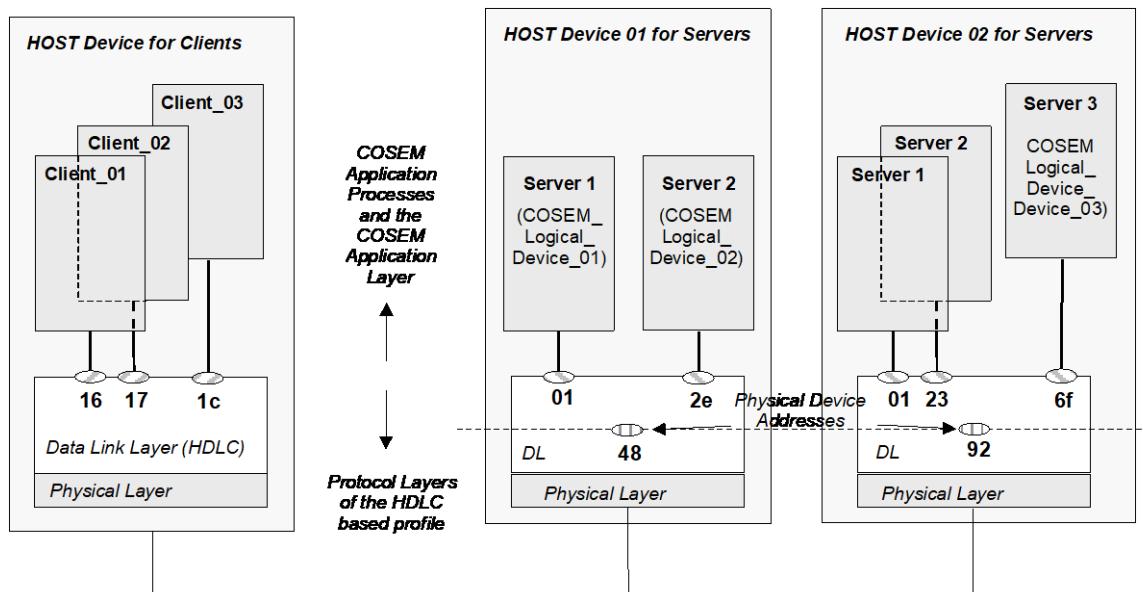
### 10.2.3 Identification and addressing scheme

The HDLC based data link layer provides services to the DLMS/COSEM AL at Data Link SAP-s, also called as the Data Link- or HDLC addresses.

On the client side, only the client AP needs to be identified. The addressing of the physical device hosting the client APs is done by the PhL (for example by using phone numbers).

On the server side, several physical devices may share a common physical line (multidrop configuration). In the case of direct connection this may be a current loop as specified in IEC 62056-21. In the case of remote connection several physical devices may share a single telephone line. Therefore both the physical devices and the logical devices hosted by the physical devices need to be identified. This is done using the HDLC addressing mechanism as described in 8.4.2:

- physical devices are identified by their lower HDLC address;
- logical devices within a physical device are identified by their upper HDLC address;
- a COSEM AA is identified by a doublet, containing the identifiers of the two APs participating in the AA.



**Figure 153 – Identification/addressing scheme in the 3-layer, CO, HDLC based communication profile**

For example, an AA between Client\_01 (HDLC address = 16) and Server 2 in Host Device 02 (HDLC address = 2392) is identified by the doublet {16, 2392}. Here, “23” is the upper HDLC address and “92” is the lower HDLC address. All values are hexadecimal. This scheme ensures that a particular COSEM AP (client or server) may support more than one AA simultaneously without ambiguity. See Figure 153.

#### 10.2.4 Supporting protocol layer services and service mapping

In this profile, the supporting protocol layer of the DLMS/COSEM AL is the HDLC based data link layer. It provides services for:

- data link layer connection management;
- connection-oriented data transfer;
- connection-less data transfer.

Figure 154 summarizes the data link layer services provided for and used by the DLMS/COSEM AL.

The DL-DATA.confirm primitive on the server side is available to support transporting long messages from the server to the client in a transparent manner to the AL. See 10.2.6.5.

In some cases, the correspondence between an AL (ASO) service invocation and the supporting data link layer service invocation is straightforward. For example, invocation of a GET.request primitive directly implies the invocation of a DL-DATA.request primitive.

In some other cases a direct service mapping cannot be established. For example, the invocation of a COSEM-OPEN.request primitive with Service\_Class == Confirmed involves a series of actions, starting with the establishment of the lower layer connection with the help of the DL-CONNECT service, and then sending out the AARQ APDU via this newly established connection using a DL-DATA.request service. Examples for service mapping are given in 9.4.

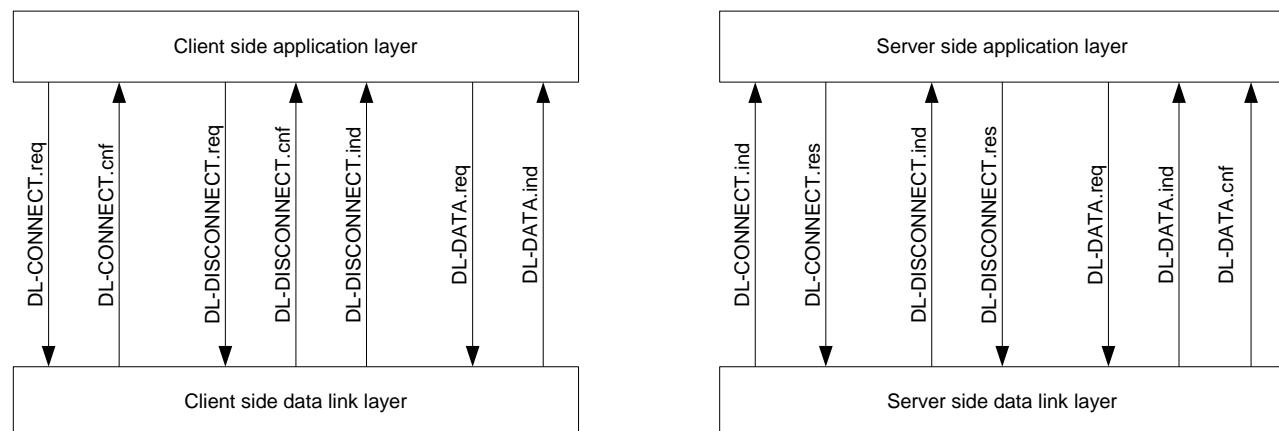


Figure 154 – Summary of data link layer services

#### 10.2.5 Communication profile specific service parameters of the DLMS/COSEM AL services

Only the COSEM-OPEN service has communication profile specific parameters, the Protocol\_Connection\_Parameters parameter. This contains the following data:

- Protocol (Profile) Identifier      3-Layer, connection-oriented, HDLC based;
- Server\_Lower\_MAC\_Address      (COSEM Physical Device Address);
- Server\_Upper\_MAC\_Address      (COSEM Logical Device Address);
- Client\_MAC\_Address;
- Server\_LLC\_Address;
- Client\_LLC\_Address.

Any server (destination) address parameter may contain special addresses (All-station, No-station, etc.). For more information, see Clause 8.

### 10.2.6 Specific considerations / constraints

#### 10.2.6.1 Confirmed and unconfirmed AAs and data transfer service invocations, frame types used

Table 98 summarizes the rules for establishing confirmed and unconfirmed AAs, the type of data transfer services available in such AAs and the HDLC frame types that carry the APDU-s. This table clearly shows one of the specific features of this profile: the Service\_Class parameter of service invocations is linked to the frame type of the supporting protocol layer:

- if the COSEM-OPEN service – see 9.3.2 – is invoked with Service\_Class == Confirmed, then the AARQ APDU is carried by an “I” frame. On the other hand, if it is invoked with Service\_Class == Unconfirmed it is carried by a “UI” frame. Therefore, in this profile, the response-allowed parameter of the xDLMS InitiateRequest APDU has no significance. See also 9.4.4.1;
- similarly, if a data transfer service .request primitive is invoked with Service\_Class == Confirmed, then the corresponding APDU is transported by an “I” frame. If it is invoked with Service\_Class == Unconfirmed then the corresponding APDU is carried by a “UI” frame. Consequently, service-class bit of the Invoke-Id-And-Priority / Long-Invoke-Id-And-Priority field – see 9.5 – is not relevant in this profile.

When the server is accessed via a gateway – see 10.10 – and the APDU is encrypted, the gateway is not able to check the response-allowed field of the xDLMS InitiateRequest APDU or the service-class bit of the Invoke-Id-And-Priority / Long-Invoke-Id-And-Priority field to determine if the APDU carries a confirmed or an unconfirmed service request.

Therefore when the 3-layer C.O. HDLC based profile is used between the gateway and server, the gateway always places the APDU received to an I frame and forwards it to the server.

The server shall not respond if it receives:

- an AARQ APDU carried by an I frame and the response-allowed field of the xDLMS InitiateRequest APDU is set to FALSE;
- an xDLMS APDU in an I frame with the service-class bit of the Invoke-Id-And-Priority / Long-Invoke-Id-And-Priority field present and set to 0;
- an AARQ or an xDLMS APDU in a UI frame.

**Table 98 – Application associations and data exchange in the 3-layer, CO, HDLC based profile**

Application association establishment				Data exchange	
Protocol connection parameters	COSEM-OPEN service class	Use	Type of established AA	Service class	Use
Id: HDLC LLC and MAC addresses	Confirmed	1/ Connect data link layer 2/ Exchange AARQ/AARE APDU-s transported in “I” frames	Confirmed	Confirmed	“I” frame
				Unconfirmed	“I” frame or “UI” frame
	Unconfirmed	Send AARQ in a “UI” frame	Unconfirmed	Confirmed (not allowed)	-
				Unconfirmed	“I” frame or “UI” frame

NOTE As described above this table, when the server is accessed via a gateway, the COSEM APDUs are always carried by I frames. The server has to check the response-allowed field of the xDLMS InitiateRequest APDU or the service-class bit of Invoke-Id-And-Priority / Long-Invoke-Id-And-Priority field as applicable to determine if the service request is confirmed or unconfirmed.

### 10.2.6.2 Correspondence between AAs and data link layer connections, releasing AAs

In this profile, a confirmed AA is bound to a supporting data link layer connection, in a one-to-one basis. Consequently:

- establishing a confirmed AA implies the establishment of a connection between the client and server data link layers;
- a confirmed AA in this profile can be non-ambiguously released by disconnecting the corresponding data link layer connection.

On the other hand, in this profile establishing an unconfirmed AA does not need any lower layer connection: consequently, once established, unconfirmed AAs with servers not supporting the ACSE A-RELEASE service (see 9.3.3 and 9.4.5) cannot be released.

### 10.2.6.3 Service parameters of the COSEM-OPEN / -RELEASE / -ABORT services

Thanks to the possibility to transparently transport higher layer related information within the SNRM and DISC HDLC frames, this profile allows the use of the optional "User\_Information" parameter of the COSEM-OPEN – see 9.3.2 – and COSEM-RELEASE – see 9.3.3 – services:

- the User\_Information parameter of a COSEM-OPEN.request primitive, if present, is inserted into the "User data subfield" of the SNRM frame, sent during the data link connection establishment;
- if the SNRM frame received by the server contains a "User data subfield", its contents is passed to the server AP via the User\_Information parameter of the COSEM-OPEN.indication primitive;
- the User\_Information parameter of a COSEM-RELEASE.request primitive, if present, is inserted into the "User data subfield" of the DISC frame, sent during disconnecting the data link connection;
- if the DISC frame received by the server contains a "User data subfield", its contents is passed to the server AP via the User\_Information parameter of the COSEM-RELEASE.indication primitive;
- the User\_Information parameter of the COSEM-RELEASE.response primitive, if present, is inserted into the "User data subfield" of the UA or HDLC frame, sent in response to the DISC frame;
- if the UA or DM frame received by the client contains "User data subfield", its contents is passed to the client AP via the User\_Information parameter of the COSEM-RELEASE.confirm primitive.

In addition, for the COSEM-ABORT .indication service primitive, the following rule applies:

- the Diagnostics parameter of the COSEM-ABORT.indication primitive – see 9.3.4 – may contain an unnumbered send status parameter. This parameter indicates whether, at the moment of the physical abort indication, the data link layer has or does not have a pending Unnumbered Information message (UI). The type and the value of this parameter is a local issue, thus it is not within the Scope of this Technical Report. See also 8.2.3.3.

### 10.2.6.4 EventNotification service and protocol

This subclause describes the communication profile specific elements of the protocol of the EventNotification service, see 9.4.6.8.

In this profile, an event is reported always by the server Management Logical Device (mandatory, reserved upper HDLC address 0x01) to the Client Management AP (mandatory, reserved HDLC address 0x01).

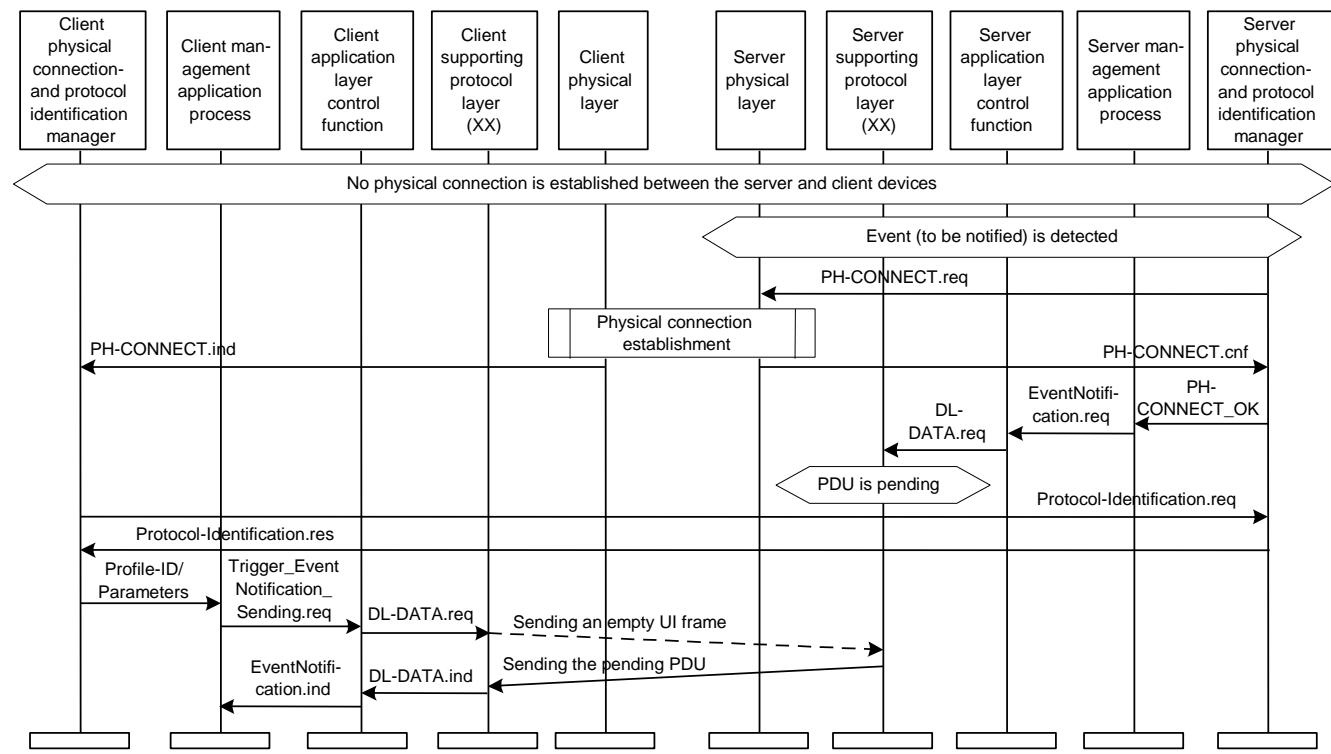
The event-notification-request APDU is sent using connectionless data services, using an UI frame, at the first opportunity, i.e. when the server side data link layer receives the right to talk. The APDU shall fit into a single HDLC frame. To be able to send out the APDU, a physical connection between the physical device hosting the server and a client device must exist, and the server side data link layer needs to receive the token from the client side data link layer.

If there is a data link connection between the client and the server when the event occurs, the server side data link layer may send out the PDU – carrying the event-notification-request APDU – following the reception of an I, a UI or an RR frame from the client. See 8.4.5.4.7.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	411/614
-----------------------	------------	-----------------------	---------

Figure 155 shows the procedure in the case, when there is no physical connection when the event occurs (but this connection to a client device can be established).

**NOTE** Physical connection cannot be established when the server has only a local interface (for example an optical port as defined in IEC 62056-21) and the HHU, running the client application is not connected, or the server has a PSTN interface, but the telephone line is not available. Handling such cases is implementation specific.



**Figure 155 – Example: EventNotification triggered by the client**

The first step is to establish this physical connection.

**NOTE** This physical connection establishment is done outside of the protocol stack.

If successful, this is reported at both sides to the physical connection manager process. At the server side, this indicates to the AP that the EventNotification.request service can be invoked now. When it is done, the server AL builds an event-notification-request APDU and invokes the connectionless DL-DATA.request primitive of the data link layer with the data parameter carrying the APDU. However, the data link layer may not be able to send this APDU, thus it is stored in the data link layer, waiting to be sent (pending).

When the client detects a successful physical connection establishment – and as there is no other reason to receive an incoming call – it supposes that this call is originated by a server intending to send the event-notification-request APDU.

At this moment, the client may not know the protocol stack used by the calling server. Therefore, it has to identify it first using the optional protocol identification service described in Clause 5. This is shown as a “Protocol-Identification.request” – “Protocol-Identification.response” message exchange in Figure 155. Following this, the client is able to instantiate the right protocol stack.

The client AP invokes then the TriggerEventNotificationSending .request primitive (see 9.3.12). Upon invocation of this primitive, the AL invokes the connectionless DL-DATA.request primitive of the data link layer with empty data, and the data link layer sends out an empty UI frame with the P/F bit set to TRUE, giving the permission to the server side data link layer to send the pending PDU.

When the client AL receives an event-notification-request APDU, it generates the EventNotification .indication primitive. The client is notified now about the event, the sequence is completed.

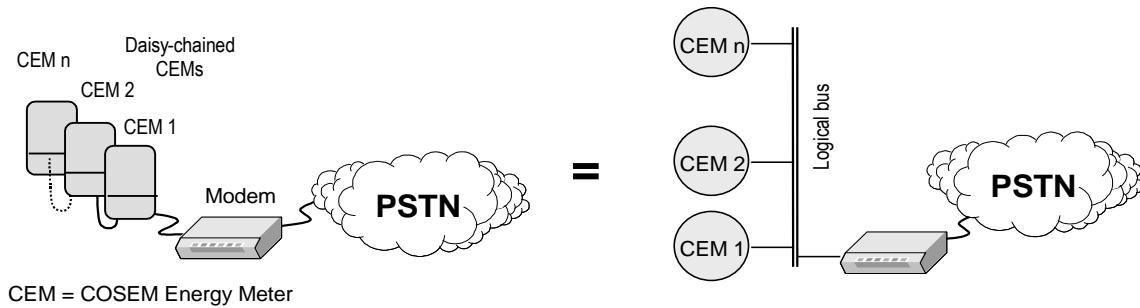
### 10.2.6.5 Transporting long messages

In this profile, the data link layer provides a method for transporting long messages in a transparent manner for the AL. This is described in 8.4.5.4.5. See also 9.1.4.4.5.

As transparent long data transfer is specified only for the direction from the server to the client, the server side supporting protocol layer provides special services for this purpose to the server AL. As these services are specific to the supporting protocol layer, no specific AL services and protocols are specified for this purpose. When the supporting protocol layer supports transparent long data transfer, the server side AL implementation may be able to manage these services.

### 10.2.6.6 Supporting multi-drop configurations

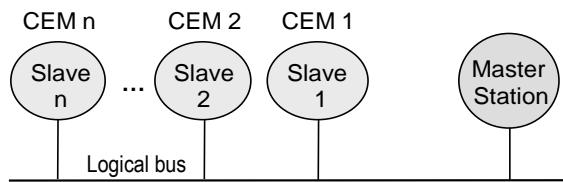
A multi-drop arrangement is often used allowing a data collection system to exchange data with multiple instances of physical devices, using a shared communication resource such as a telephone modem. Various physical arrangements are available such as a star, daisy chain or a bus topology. These arrangements can be modelled with a logical bus, to which the devices and the shared resource are connected, see Figure 156.



**Figure 156 – Multi-drop configuration and its model**

A protocol controlling access to the shared resource is not available to prevent collisions on the bus so access to it must be controlled by external rules. In most cases, a Master-Slave arrangement is used, where the devices are the Slaves. Slave devices have no right to send messages without first receiving an explicit permission from the Master.

In DLMS/COSEM, data exchange takes place based on the client/server model. Physical devices are modelled as a set of logical devices, acting as servers, providing responses to requests. The Master Station of a multi-drop configuration is located at the other end of the communication channel and it acts as the client, sending requests and expecting responses.



**Figure 157 – Master/ Slave operation on the multi-drop bus**

The client may send requests at the same time to multiple servers provided that no responses are expected (multi-cast or broadcast). If the client expects a response, it must first send a request to a single server which then gives it permission to talk. The client then has to then to for the response to its message from that server before it can send a request to another server that will then give it permission to talk to it. Arbitration of access to the common bus is thus controlled through a time-multiplexing process.

Messages from the client to the servers must contain addressing information. In this profile, the HDLC addressing scheme provides the necessary addressing information. Where a multi-drop arrangement is used, the HDLC address is split to two parts to address logical devices within the physical device:

- the lower HDLC address to address physical devices and
- the upper HDLC address.

Both the lower and the upper address may contain a broadcast address. For details, see 8.4.2.

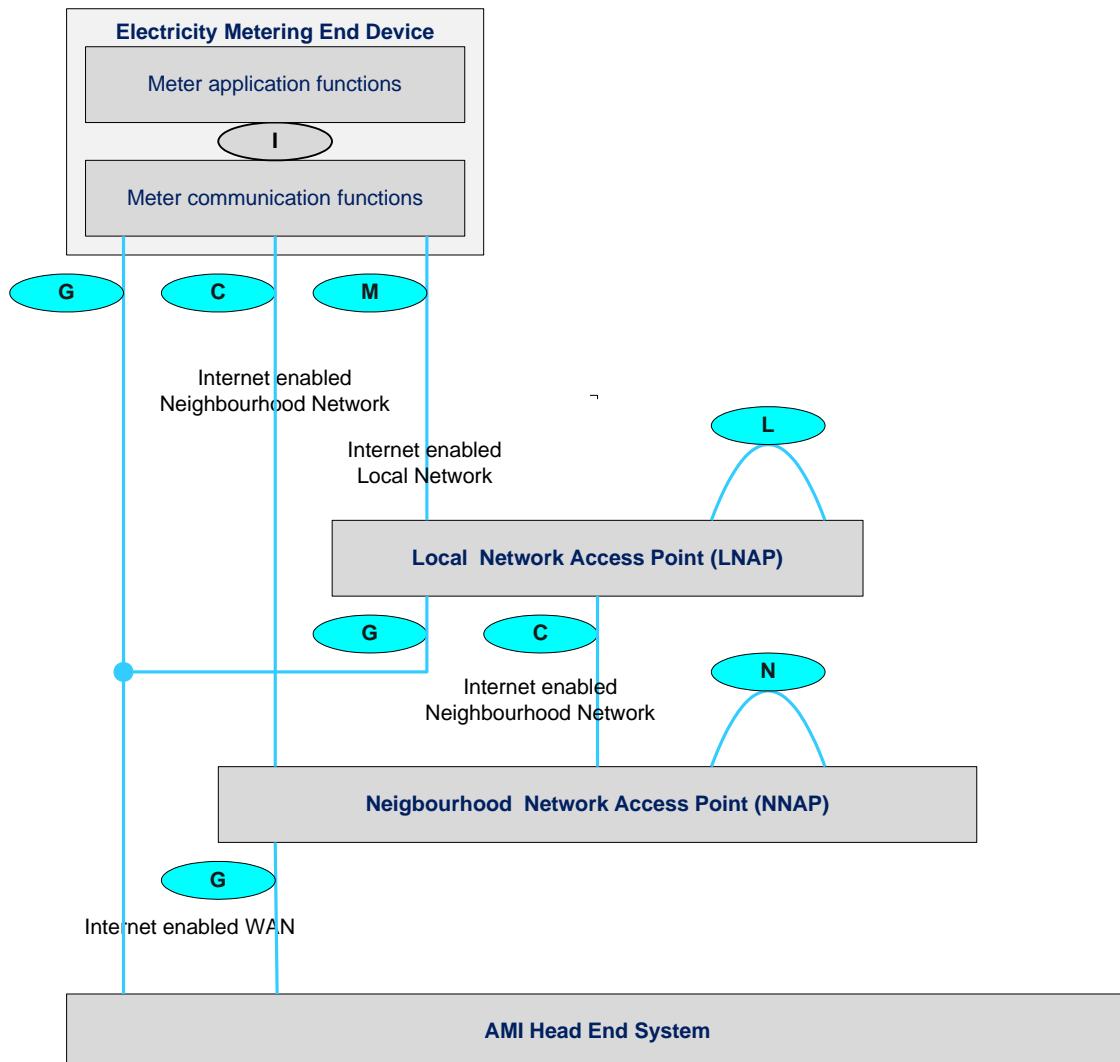
To enable the reporting of events, a server may initiate a connection to the client, using the unsolicited EventNotification / InformationReport services. As events in several or all servers connected to a multidrop may occur simultaneously they may initiate a call to the client simultaneously (for example in the case of a power failure). For such cases, two potential problems have to be handled:

- collision on the logical bus:  
For the reasons explained above, several physical devices may try to access the shared resource simultaneously (for example sending AT commands to the modem). Such situations must be handled by the manufacturers;
- identification of the originator of the event report:  
this is possible by using the CALLING Physical Device Address, as described in 8.4.5.4.8.

## 10.3 The TCP-UDP/IP based communication profiles (COSEM\_on\_IP)

### 10.3.1 Targeted communication environments

The TCP-UDP/IP based communication profiles are suitable for remote data exchange with devices via IP enabled networks such as Wide Area Networks, Neighbourhood Networks or Local Networks. This is shown in Figure 158.



**Figure 158 – Communication architecture**

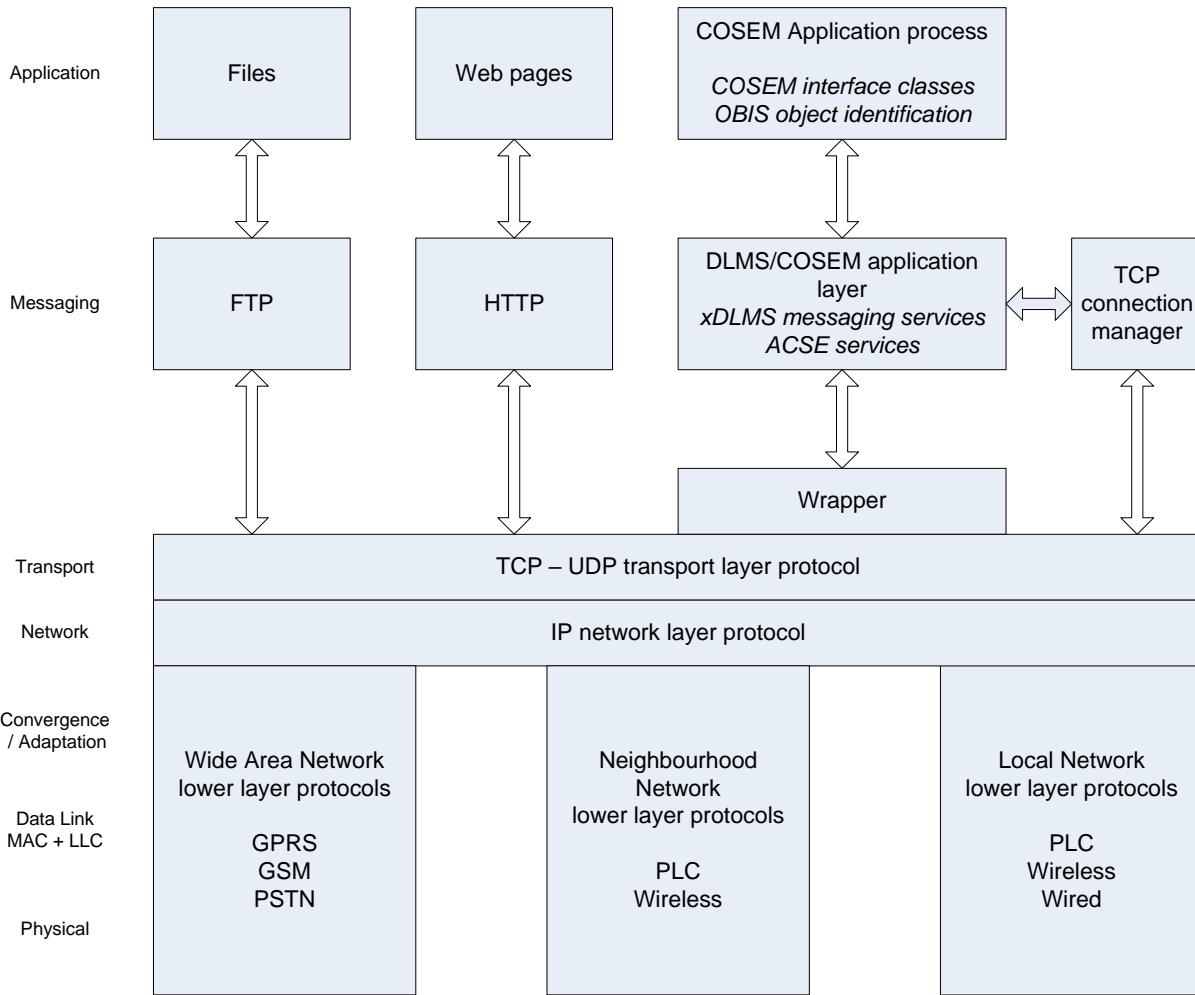
### 10.3.2 The structure of the profile(s)

The COSEM TCP-UDP/IP based communication profiles consist of five protocol layers:

- the DLMS/COSEM Application layer, specified in Clause 9;
- the DLMS/COSEM transport layer, specified in Clause 7;
- a network layer: the Internet Protocol (IPv4 or IPv6);
- a data link layer: any data link protocol supporting the network layer;
- a physical layer: any PhL supported by the data link layer chosen.

The DLMS/COSEM AL uses the services of one of the TLs (TCP or UDP) via a wrapper, which, in their turn, use the services of the IPv4 or IPv6 network layer to communicate with other nodes connected to this abstract network. The DLMS/COSEM AL in this environment can be considered as another Internet standard application protocol, which may co-exist with other Internet application protocols, like FTP, HTTP etc. See Figure 26.

The TCP-UDP/IP layers are implemented on a wide variety of real networks, which, just with the help of this IP Network abstraction, can be seamlessly interconnected to form Intra- and Internets using any set of lower layers supporting the Internet Protocol.



**Figure 159 – Examples for lower-layer protocols in the TCP-UDP/IP based profile(s)**

Below the IP layer, a range of lower layers can be used. One of the reasons of the success of the Internet protocols is just their federating force. Practically any data networks, including Wide Area Networks such as GPRS, ISDN, ATM and Frame Relay, circuit switched PSTN and GSM networks (dial-up IP), Local Area Networks, such as Ethernet, neighbourhood networks and local networks using power line carrier or wireless protocols, etc. support TCP-UDP/IP networking.

Figure 159 shows a set of examples – far from being complete – for such communication networks and for the lower layer protocols used in these networks. Using the TCP-UDP/IP profile, DLMS/COSEM can be used practically on any existing communication network.

### 10.3.3 Identification and addressing scheme

Although real-world devices even in the Internet environment are connected to real-world physical networks, at a higher abstraction (and protocol) level it can be considered as if these devices would be connected to a virtual – IP – network. On this virtual network, each device has a unique address, called IP address, which non-ambiguously identifies the device on this network.

Any device connected to this virtual IP network can send message(s) to any other connected device(s) using only the IP address to designate the destination device, without being concerned about the complexity of the whole physical network. Specific characteristics – the data transmission medium, the media access strategy, and the specific data-link addressing / identification scheme – of the particular

physical network(s) participating in the route between the source and the destination device are hidden for the sender device. These elements are handled by intermediate network devices, called routers.

Therefore, in the TCP-UDP/IP based profiles COSEM physical devices are non-ambiguously identified by their network – IP – address.

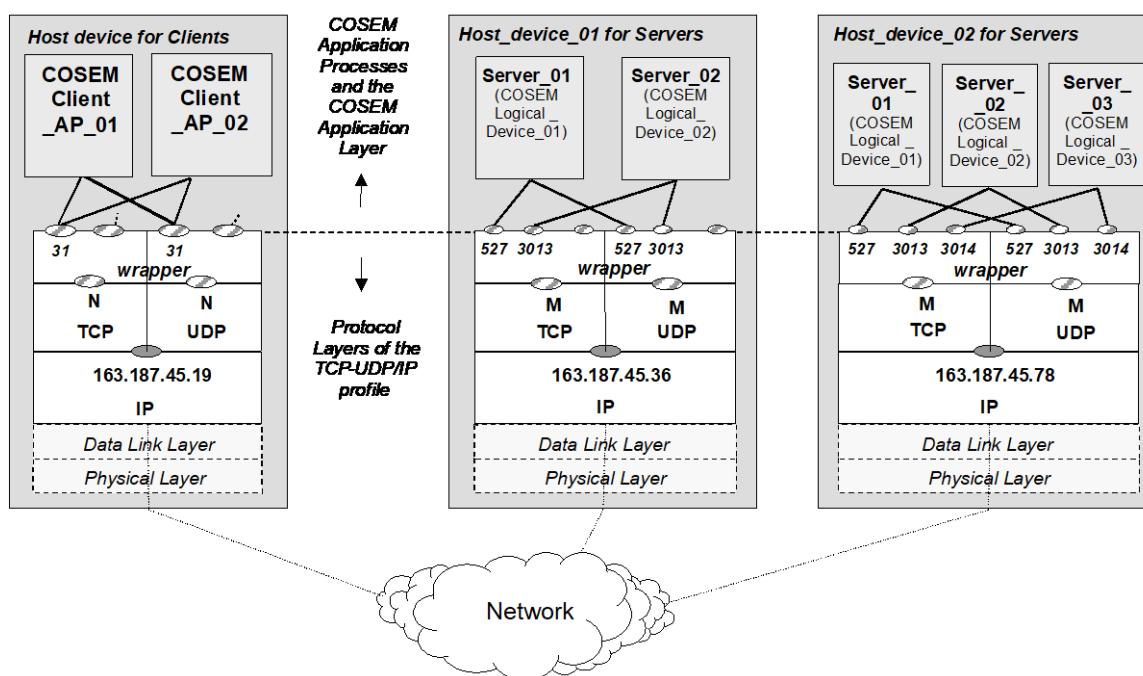
The identification of COSEM client AP and server APs requires an additional address.

Both TCP and UDP provide additional addressing capability at the transport level, called *port*, to distinguish between applications. The AL is listening only on one TCP or UDP port for exchanging messages between any client and server APs. As in a single physical device several client or server APs may be present, an additional addressing capability is needed. This is provided by the wrapper sublayer, see Clause 7. The wrapper provides an identifier – wPort – similar to the TCP or UDP port numbers, but on the top of these layers. A particular COSEM client AP and/or a particular COSEM logical device in the same physical device can be thus identified by its wPort number.

In summary, in the TCP-UDP/IP based profiles the following identification rules apply:

- COSEM physical devices are identified by their IP address;
- the DLMS/COSEM AL is listening only on one UDP or TCP port. See 7.2;
- COSEM logical devices and client APs within their respective host physical devices are identified by their wPort numbers. Reserved wPort numbers are specified in Clause 7;
- lower layer addresses (SAP-s) are not considered (hidden).

AAs are identified by the identifiers of the two end-points as described above. Figure 160 shows an example.



**Figure 160 – Identification / addressing scheme in the TCP-UDP/IP based profile(s)**

AAs established between the client AP\_01 and Logical\_Device\_01 in Host\_device\_01 (AA 1) and Logical\_Device\_02 in Host\_Device\_02 (AA2) respectively are identified by:

$$\begin{aligned} \text{AA 1: } & \{ (163.187.45.19, T_N, 31) (163.187.45.36, T_M, 527) \} \\ \text{AA 2: } & \{ (163.187.45.19, T_N, 31) (163.187.45.78, T_M, 3013) \} \end{aligned}$$

NOTE 1 T\_N and T\_M mean the TCP port used for DLMS/COSEM in the client host device and the server host devices respectively. For DLMS/COSEM, the following port numbers have been registered by the IANA. See <http://www.iana.org/assignments/port-numbers>:

- dlms/cosem 4059/TCP DLMS/COSEM;
- dlms/cosem 4059/UDP DLMS/COSEM.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	417/614
-----------------------	------------	-----------------------	---------

NOTE 2 In these two AAs the client side end-point identifiers are the same. However, the server side end-point identifiers are different, so the two AAs are identified unambiguously and therefore they can be used simultaneously.

NOTE 3 In these examples, IPv4 addresses are used.

#### 10.3.4 Supporting protocol layer services and service mapping

As specified in Clause 7, the COSEM TCP TL provides the following services to its service users:

- Connection management services, provided for the TCP connection manager AP:
  - TCP-CONNECT.request, .indication, .response, .confirm;
  - TCP-DISCONNECT.request, .indication, .response, .confirm;
- Data exchange services, provided for the DLMS/COSEM AL; these services can be used only when the TCP connection is established:
  - TCP-DATA – .request, .indication, (. confirm).

The TCP TL also provides a TCP-ABORT service to the service user DLMS/COSEM AL to indicate the disconnection/disruption of the TCP layer connection.

The UDP TL provides only one service to the service user DLMS/COSEM AL: a connection-less, best effort data delivery service:

- UDP-DATA – .request, .indication, (.confirm)

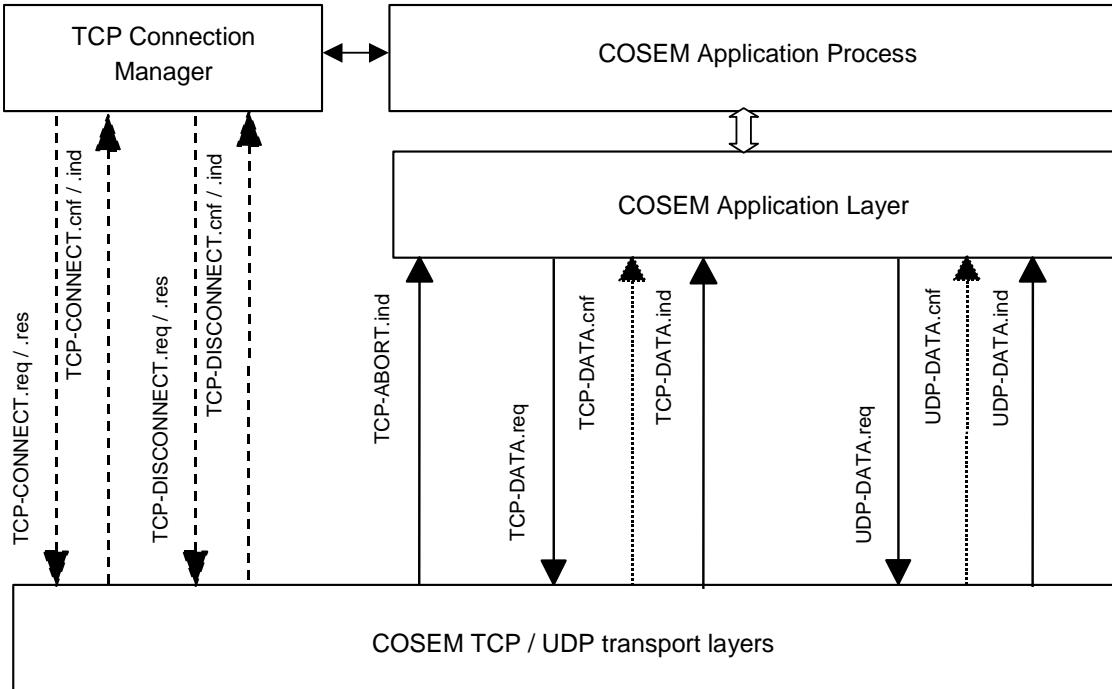
NOTE A TCP.confirm / UDP .confirm service primitive is optionally available.

Figure 161 summarizes these services.

For connection management, the COSEM TCP TL provides the full set of the TCP-CONNECT and TCP-DISCONNECT services, both at the client- and at the server sides. The purpose of this is to allow also the server to establish and release TCP connections. See also 10.3.6.7. As in all COSEM profiles, AA establishment and release is initiated by the client AP in these profiles as well.

The user of these services is not the DLMS/COSEM AL, but the TCP Connection Manager AP. This process is implementation dependent; therefore it is out of the Scope of this Technical Report. The only requirements with regard to this process are:

- the TCP connection manager process shall be able to establish the supporting TCP connection without the intervention of the COSEM client- or server AP(s);
- the COSEM client- and server APs must be able to retrieve the TCP and IP portion of the Protocol\_Connection\_Parameters parameter from the TCP connection manager before sending / receiving a COSEM-OPEN.request / .indication.

**Figure 161 – Summary of TCP / UDP layer services**

For data exchange, both the client- and the server ALs use the complete set of the service primitives provided by the COSEM TCP-UDP TLs.

The correspondence between an AL (ASO) service invocation and the supporting COSEM TCP-UDP layer service invocation is given in Clause 7.

### 10.3.5 Communication profile specific service parameters of the DLMS/COSEM AL services

Only the COSEM-OPEN service has communication profile specific parameters, the Protocol\_Connection\_Parameters parameter. This contains the following data:

- Protocol (Profile) Identifier      TCP/IP or UDP/IP;
- Server\_IP\_Address                COSEM Physical Device Address;
- Server\_TCP\_or\_UDP\_Port         The TCP or UDP port used for DLMS/COSEM, see 7.2;
- Server\_Wrapper\_Port             COSEM Logical Device Address;
- Client\_IP\_Address                COSEM Client's Physical Device Address;
- Client\_TCP\_or\_UDP\_Port         The TCP or UDP port used for DLMS/COSEM, see 7.2;
- Client\_Wrapper\_Port             COSEM application process (type) identifier.

Any server address parameter may contain special addresses (All-station, No-station, etc...). For more information, see Clause 7.

### 10.3.6 Specific considerations / constraints

#### 10.3.6.1 Confirmed and unconfirmed AAs and data transfer service invocations, packet types used

Table 99 shows the rules for establishing confirmed and unconfirmed AAs, the type of data transfer services available in such AAs and the TL packet types used for carrying APDU-s. In this table, grey areas represent cases, which are out of the normal operating conditions: either not allowed or have no useful purpose.

According to these:

- it is not allowed to establish an unconfirmed AA using the TCP/IP protocol. It is prevented by the Client AL, which locally and negatively confirms COSEM-OPEN.request primitive invocations trying to do that;
- it is not allowed to request an xDLMS service in a confirmed way (Service\_Class = Confirmed) within an unconfirmed AA, established on the top of the UDP layer. This is also prevented by the Client AL. Servers, receiving such APDUs shall simply discard them, or, shall send back a confirmedServiceError APDU or, if the feature is implemented, send back the optional exception-response APDU.

**Table 99 – Application associations and data exchange in the TCP-UDP/IP based profile**

Application association establishment				Data exchange	
Protocol connection parameters	COSEM-OPEN service class	Use	Type of established application association	Service class	Use
Id: TCP/IP TCP port numbers, IP addresses	Confirmed	1/ Connect TCP layer 2/ Exchange AARQ/AARE APDU-s transported in TCP packets	Confirmed	Confirmed	TCP packet
				Unconfirmed	TCP packet
	Unconfirmed	Local negative confirmation	None	-	-
				-	-
Id: UDP/IP UDP port numbers, IP addresses	Confirmed	Exchange AARQ/AARE APDU-s transported in UDP datagrams	Confirmed	Confirmed	UDP datagram
				Unconfirmed	UDP datagram
	Unconfirmed	Send AARQ in a UDP datagram	Unconfirmed	Confirmed (not allowed)	-
				Unconfirmed	UDP datagram

In the TCP-UDP/IP based profiles, the Service\_Class parameter of the COSEM-OPEN service is linked to the response-allowed parameter of the xDLMS InitiateRequest APDU. If the COSEM-OPEN service is invoked with Service\_Class == Confirmed, the response-allowed parameter shall be set to TRUE. The server is expected to respond. If it is invoked with Service\_Class == Unconfirmed, the response-allowed parameter shall be set to FALSE. The server shall not send back a response.

The Service\_Class parameter of the GET, SET and ACTION services is linked to service-class bit of the Invoke-Id-And-Priority byte. If the service is invoked with Service\_Class = Confirmed, service-class bit shall be set to 1, otherwise it shall be set to 0.

#### 10.3.6.2 Releasing application associations: using RLRQ/RLRE is mandatory

In the TCP-UDP/IP based profile, using the A-RELEASE services of the ACSE – by invoking the COSEM-RELEASE.request primitive with Use\_RLRQ\_RE == TRUE – is mandatory for the following reasons:

- according to the identification / addressing scheme used in this profile, an AA is identified by two triplets, including the IP address, the TCP (or UDP) port number and the wPort number. In other words, all AAs within this profile are established using only one TCP (or UDP) port. This means, that disconnecting the TCP connection (this way of releasing AA must also be supported) would release all AAs established. Using the RLRQ/RLRE APDU-s allows to release confirmed AAs in a selective way;

- it is allowed to establish both confirmed and unconfirmed AAs on the connectionless UDP TL. The only way to release such associations is the use of the RLRQ/RLRE services.

**NOTE** In fact, using the RLRQ/RLRE APDU-s is specified as optional only to keep backward compatibility with the third edition of the Green Book (2002), which did not include this possibility.

#### 10.3.6.3 Service parameters of the COSEM-OPEN / -RELEASE / -ABORT services

The optional User\_Information parameters of the COSEM-OPEN / -RELEASE services are not supported in this communication profile.

#### 10.3.6.4 xDLMS request/response type services

No specific features / constraints apply related to the use of request/response type services.

#### 10.3.6.5 The EventNotification Service and the TriggerEventNotificationSending service

This subclause describes the communication profile specific elements of the protocol of the EventNotification service, see 9.4.6.8.

The TriggerEventNotificationSending service is not used in either the TCP or UDP profiles as they allow the sending of data in an unsolicited manner.

The event-notification-request APDU may be sent either using the connectionless data services of the COSEM UDP-based TL or by the connection-oriented data services of the COSEM TCP-based TL. In this case, a TCP connection has to be built first by the TCP Connection Manager process.

The optional Application\_Addresses parameter is present only when the EventNotification.request service is invoked outside of an established AA.

#### 10.3.6.6 Transporting long messages

The data field of the wrapper sublayer shall always carry a complete xDLMS APDU. If the APDU is too long to be accommodated, then application layer block transfer can be used.

#### 10.3.6.7 Allowing COSEM servers to establish the TCP connection

In DLMS/COSEM, supporting protocol layer connections are generally established during AA establishment following the invocation of the COSEM-OPEN.request primitive by the client AP (the PhL connection must be already established before invoking the COSEM-OPEN.request primitive). Therefore linking the process of establishing an AA and connecting the supporting protocol layer is just natural.

However, in some cases it would be useful if the server could also initiate the connection of the TCP layer. This is particularly interesting in the TCP-UDP/IP based profile, when the server does not have a public IP address. In this case, as the Client does not "see" the physical device hosting the server(s), it is not able to establish the required TCP layer connection.

In order to allow the server to establish the TCP layer connection, the full set of service primitives of the TCP-CONNECT service is available both on the client and the server side.

**NOTE** These services are used by the TCP connection manager, not by the AL.

#### 10.3.6.8 The COSEM TCP-UDP/IP profile and real-world IP networks

Clauses 7 and 9 specify all elements necessary to use DLMS/COSEM over the Internet, using the DLMS/COSEM TCP-UDP/IP based profile.

On real Internet networks, there are other elements, which need to be considered. For example, in this Technical Report it is specified, that physical devices hosting COSEM APs are identified with an IP address, but it is not specified, how to obtain such an IP address. As these elements are not specific to DLMS/COSEM, they are not in the Scope of this Technical Report.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	421/614
-----------------------	------------	-----------------------	---------

**10.4 The CoAP based communication profile (DLMS/COSEM\_on\_CoAP)****10.4.1 Targeted communications environments**

The CoAP based communication profile, as the TCP-UDP/IP based communication profiles, are suitable for remote data exchange with devices via IP enabled networks such as Wide Area Networks, Neighbourhood Networks or Local Networks. For reference see Figure 158.

The CoAP based communication profile is particularly relevant for constrained devices and for data exchange over constrained (e.g., lossy) networks.

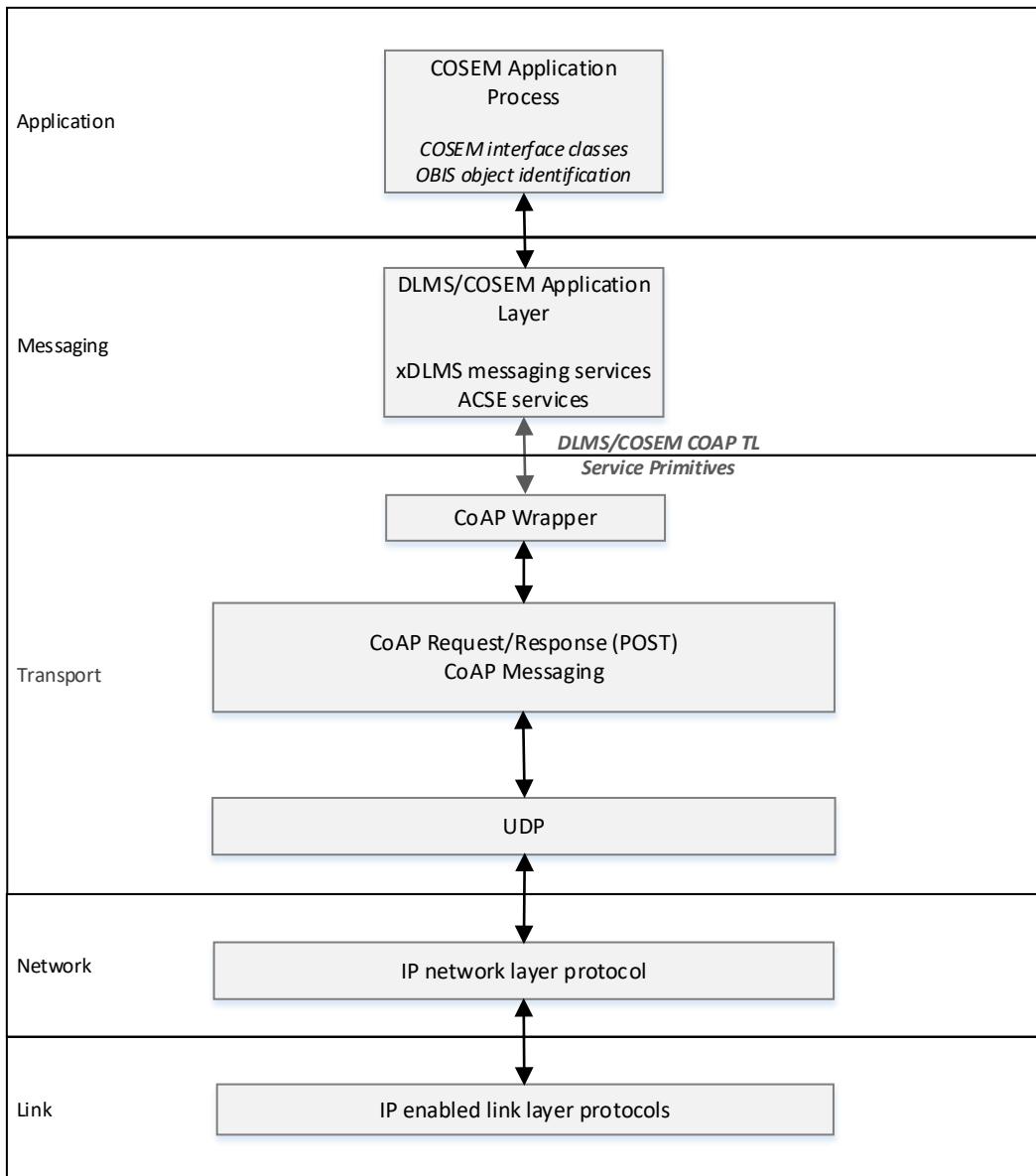
**10.4.2 The structure of the CoAP communication profile**

The DLMS/COSEM CoAP based communication profile consists of five protocol layers:

- the DLMS/COSEM Application layer as specified in 9;
- the DLMS/COSEM CoAP transport layer embedding the CoAP wrapper layer, the CoAP transport protocol layer and the UDP layer, as specified in 7.3;
- a network layer: the Internet Protocol (IPv4 or IPv6);
- a data link layer: any data link protocol supporting the network layer;
- a physical layer: any PhL supported by the data link layer chosen.

The DLMS/COSEM AL accesses the CoAP request/response layer via a DLMS/COSEM CoAP wrapper layer that provides the DLMS/COSEM CoAP-DATA service primitives to the DLMS/COSEM AL. The DLMS/COSEM CoAP wrapper layer interfaces with the CoAP block transfer capable CoAP protocol layers implementing the CoAP request/response layer and CoAP messaging layer functions as defined in their base standards RFC 7252 and RFC 7959. The CoAP protocol layers employ a standard UDP transport layer running on top of the IPv4 or IPv6 network layer to communicate with other nodes connected to this abstract IP network. The DLMS/COSEM CoAP wrapper interfaces the CoAP request/response transfer services through usage of the POST method. For reference see Figure 162.

When used in this profile, DLMS/COSEM AL operates as an Internet standard serialization and security standard. The use of the CoAP transport mechanism aligns the DLMS/COSEM application with other Internet application protocols.



**Figure 162 – The DLMS/COSEM CoAP communication profile**

CoAP provides both unreliable and reliable transfer services. The unreliable transfer service is based on non-confirmable CoAP messages. The reliable transfer service is based on confirmable CoAP messages with a retry mechanism. The DLMS/COSEM communication profile supports both the reliable and the unreliable operation mode.

CoAP provides efficient and reliable transfer of request/response transactions through piggybacking of responses on CoAP acknowledgement messages. The reliable CoAP transport operation therefore saves resources in the network and in the client and server endpoints while offering full reliability.

In the DLMS/COSEM CoAP communication profile, the DLMS/COSEM application layer services are bound to the CoAP request/response layer through the CoAP wrapper. Through this binding, the DLMS/COSEM CoAP TL takes advantage of the reliable CoAP request/response transfer, which is particularly suitable for constrained nodes communicating over constrained (e.g., low power, lossy) networks.

The data field of the CoAP-DATA service primitives carries an APDU. For both the reliable and the unreliable CoAP transport service, CoAP block transfer serves to transfer large APDUs surpassing the MTU size without requiring IP layer fragmentation. DLMS/COSEM application layer block transfer mechanism

shall be used for DLMS/COSEM layer segmentation large APDU payloads surpassing the receiver\_max\_pdu\_size.

#### 10.4.3 Identification and addressing

##### 10.4.3.1 General

A DLMS/COSEM application layer using the DLMS/COSEM CoAP communication profile is uniquely identifiable via its CoAP URI. The CoAP URI is decomposed into the Uri-Host, the Uri-Port and the Uri-Path parameters. The latter may be the unspecified "empty" path. A specific Uri-Path is carried in the Uri-Path option within the CoAP messaging layer.

##### 10.4.3.2 Uri-Path

The Uri-Path of a DLMS client CoAP Server endpoint to which a DataNotification APDU shall be sent, is specified as part of the push destination specified in the "Push setup" objects, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.4.8.2.

The Uri\_Path of a DLMS client CoAP server endpoint to which an EventNotification is to be sent shall be included in the Application\_Addresses parameter of the EventNotification service primitive when the EventNotification is sent outside of an established AA. See 9.3.11.

The Uri-Path of a DLMS server may be retrieved from the "CoAP setup" object, see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.9.8.

##### 10.4.3.3 IP address assignment

As the DLMS/COSEM CoAP communication profile is based on UDP/IP transport (which is connectionless) no transport layer connection establishment management is required between the entities prior to communication. Similarly, the IP addresses and ephemeral UDP ports may dynamically change during the communication<sup>1</sup>, provided that the DLMS/COSEM AL is able to identify the sending DLMS AE by ways other than by means of the received UDP/IP ports and addresses<sup>2</sup>.

NOTE 1 The IP addresses and the UDP ephemeral ports cannot change within an individual CoAP request/response transaction.

NOTE 2 The client SAP and server SAP in any case is part of the identification.

A DLMS AE is uniquely identifiable via its system title Sys-T, whether it be a client, a server or a third party that can access servers via clients. For DLMS AEs communicating via the DLMS/COSEM CoAP communication profile, the system titles are exchanged between the DLMS entities at the following occasions:

- in the case of explicitly established AAs, during AA establishment using the COSEM-OPEN service;
- by writing the *client\_system\_title* attribute and by reading the *server\_system\_title* attribute of "Security setup" objects, see DLMS UA 1000-1 Ed. 15:2021, 4.4.7. This approach is especially relevant for pre-established AAs;
- at exchange of APDUs encoded using general-ciphering (originator and recipient system title) or general-glo-ciphering (originator system title).

Whether an AA is explicitly established or pre-established, it may be statically bound to IP addresses and UDP ports. In such a situation the DLMS/COSEM application layer may maintain a static binding between the system titles of the participating DLMS AEs and the statically assigned IP addresses and UDP ports<sup>3</sup>.

Note 3 In this situation, the DLMS/COSEM application layer may rely on the IP addresses and the UDP ports for identification in all communication with the underlying CoAP transport layer. I.e., through the CoAP-DATA service primitives.

Pre-established or long-living AAs between two DLMS/COSEM AEs may tolerate dynamic IP address assignment in the CoAP transport layer. This requires that the DLMS/COSEM application layer is able to discover the IP address that a peer DLMS/COSEM application layer is now reachable on, which is identified via its system title. This may be achieved, by pushing connectivity data when they change.

In many situations dynamic IP address assignment is applicable for the DLMS/COSEM application layer CoAP endpoints of physical devices hosting DLMS servers, whereas the IP address of the DLMS client application layer CoAP endpoint is statically assigned.

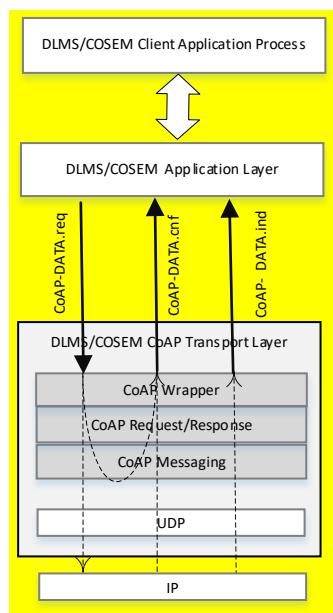
In a situation where the DLMS client application layer receives data carried in a Data-Notification APDU from a DLMS server application layer using dynamic IP address assignment, the identity of the DLMS/COSEM AE can be determined from its system title provided general-ciphering, general-glo-ciphering or general-signing is used. The DLMS/COSEM AE system title is given in originator-system-title given in the APDU.

#### 10.4.4 Supporting layer services and service mapping

The DLMS/COSEM CoAP transport layer within the DLMS/COSEM communication profile operates on UDP transport and is connection-less.

The DLMS/COSEM CoAP transport layer provides data transfer services through three data transfer services primitives, see Figure 163:

- 1) the CoAP-DATA.request primitive provided to the DLMS/COSEM AL for transfer of complete request or response APDUs;
- 2) the CoAP-DATA.indication primitive through which the DLMS/COSEM CoAP transport layer conveys received APDUs to the DLMS/COSEM AL;
- 3) the CoAP-DATA.confirm primitive through which the DLMS/COSEM CoAP transport layer conveys the following back to the DLMS/COSEM AL:
  - a) CoAP transport level confirmations from peer DLMS/COSEM CoAP transport layer endpoint of the receipt of a transferred APDU;
  - b) inability of the CoAP transport layer to transfer an APDU to peer DLMS/COSEM CoAP transport layer endpoint.



**Figure 163 – CoAP transport layer primitives**

A mapping of the DLMS/COSEM CoAP transport layer primitives to DLMS AL service primitives provided by the Association Control Service Element (ACSE) and the xDLMS Application Service Element (xDLMS ASE) is shown in Figure 164 and Figure 165.

#### 10.4.5 Communication profile specific service parameters of the DLMS/COSEM AL services

The Protocol\_Connection\_Parameters given in the COSEM-OPEN primitive contains the following data:

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	425/614
-----------------------	------------	-----------------------	---------

Protocol (profile) Identifier	CoAP
Transport_mode	Indicates reliable or unreliable CoAP transport operation
Server_IP_address	IP address of the physical device hosting a DLMS server
Server_UDP_port	UDP port of the physical device hosting a DLMS server's CoAP server end-point
Server_Uri-path	CoAP Uri-path identifying the DLMS/COSEM AL of the physical device hosting a DLMS server
Server_SAP	Service Access Point identifier for the COSEM server logical device (application process)
Client_IP_address	DLMS client's IP address. It may be left unspecified
Client_UDP_port	DLMS client's UDP port. It may be left unspecified
Client_SAP	Service Access Point identifier for the COSEM client logical device (application process)

For reference see 10.4.6.2.

#### 10.4.6 Specific considerations / constraints

##### 10.4.6.1 Confirmed and unconfirmed AAs and data transfer service invocations

Table 100 shows the rules for establishing confirmed and unconfirmed AAs using the COSEM-OPEN service and the type of data transfer services available in such AAs.

Requesting an xDLMS service in a confirmed way is not allowed within an unconfirmed AA on the top of the DLMS/COSEM CoAP transport layer. This is also prevented by the requesting DLMS AL. Receiving entities shall simply discard such APDUs, or when applicable shall send back a confirmedServiceError APDU or send back the optional exception-response APDU.

Multi- or broadcasting is not supported by the reliable CoAP transport operation.

**Table 100 – Application associations and data exchange in the CoAP based communication profile**

		Application establishment		Data exchange		
CoAP transport operations mode	COSEM-OPEN service class	Type of established application association		Service Class	Destination scope	CoAP Messages
Reliable	Confirmed	Confirmed	Confirmed	Unicast	CON	
			Confirmed	Multicast (not supported)	-	
			Unconfirmed	Unicast	CON	
			Unconfirmed	Multicast (not supported)	-	
	Unconfirmed	Unconfirmed	Confirmed (not allowed)	-	-	
			Unconfirmed	Unicast	CON	
			Unconfirmed	Multicast (not supported)	-	

	Application establishment		Data exchange		
Unreliable	Confirmed	Confirmed	Confirmed	Unicast	NON
			Confirmed	Multicast (not supported)	-
			Unconfirmed	Unicast	NON
			Unconfirmed	Multicast	NON
	Unconfirmed	Unconfirmed	Confirmed (not allowed)	-	-
			Unconfirmed	Unicast	NON
			Unconfirmed	Multicast	NON

#### 10.4.6.2 Application association establishment and release

The CoAP transport layer is connectionless and does not require transport session establishment. Provided underlying physical connectivity with IP/UDP transport ability is available, application association is established through the CoAP transport layer primitives for request/response message exchange, see Figure 164.

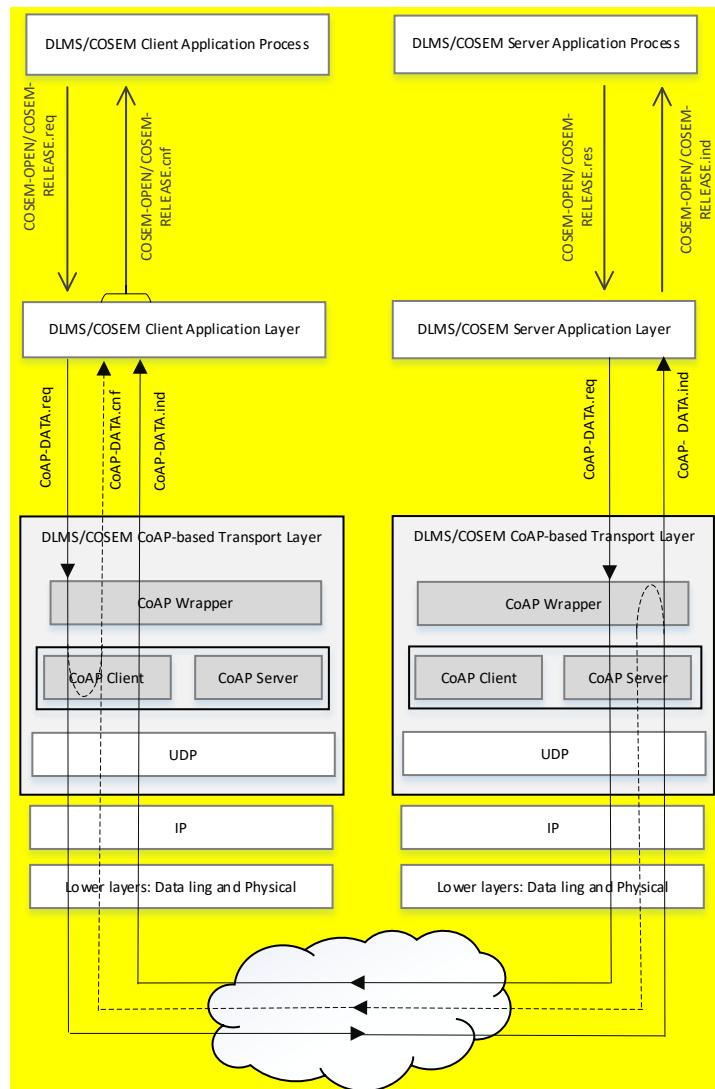


Figure 164 – Mapping the DLMS ACSE service primitives to the CoAP-DATA service primitives

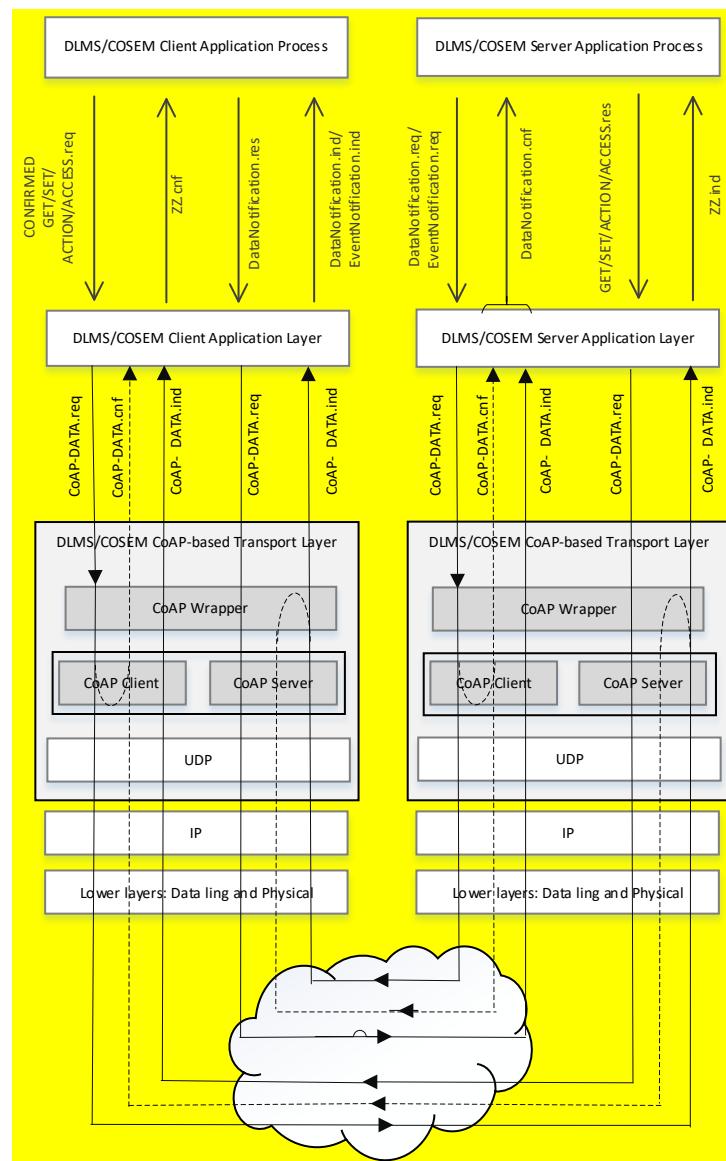
The CoAP URI parameters of the CoAP endpoint of the DLMS server shall be specified by the Application Process in the Protocol\_Connection\_Parameters of the COSEM-OPEN request. The URI-path parameter is made available to the Application Process by configuration management.

The CoAP transport layer is connectionless and therefore an AA using the DLMS/COSEM CoAP communication profile has no binding to any lower layer connection management, the only way to release association in the DLMS/COSEM over CoAP communication profile is by the use of the RLRQ/RLRE services.

As an AA using the DLMS/COSEM CoAP communication profile has no binding to any lower layer connection management, the DLMS/COSEM-ABORT primitive is not generated based on lower layer connection management state when using the DLMS/COSEM over CoAP communication profile.

#### 10.4.6.3 DLMS/COSEM data transfer service invocations

The mapping of the xDLMS application layer primitives provided by the xDLMS Application Service Element (xDLMS ASE) to the DLMS/COSEM CoAP TL primitives is indicated in Figure 165.



**Figure 165 – Mapping of the xDLMS ASE service primitives to the CoAP-DATA service primitives**

## 10.5 The S-FSK PLC profile

### 10.5.1 Terms and definitions relevant for the S-FSK PLC profile

#### 10.5.1.1

##### **initiator**

user-element of a client System Management Application Entity (SMAE). It uses the CIASE and xDLMS ASE and it is identified by its system title [IEC 61334-4-511:2000, 3.8.1 modified]

#### 10.4.1.2

##### **active initiator**

initiator, which issues or has last issued a CIASE Register request when the server is in the unconfigured state [IEC 61334-4-511:2000, 3.9.1]

#### 10.4.1.3

##### **new system**

server system, which is in the unconfigured state: its MAC address equals "NEW-address" [IEC 61334-4-511:2000, 3.9.3]

#### 10.4.1.4

##### **new system title**

system-title of a new system [IEC 61334-4-511:2000, 3.9.4]

NOTE This is the system title of a system, which is in the new state.

#### 10.4.1.5

##### **registered system**

server system, which has an individual, valid MAC address (therefore, different from "NEW Address", see IEC 61334-5-1:2001: Medium Access Control) [IEC 61334-4-511:2000, 3.9.5]

#### 10.4.1.6

##### **reporting system**

server system, which issues a DiscoverReport [IEC 61334-4-511:2000, 3.9.6 modified]

#### 10.4.1.7

##### **sub-slot**

the time needed to transmit two bytes by the physical layer

NOTE Timeslots are divided to sub-slots in the RepeaterCall mode of the physical layer.

#### 10.4.1.8

##### **timeslot**

the time needed to transmit a physical frame

NOTE As specified in IEC 61334-5-1:2001, 3.3.1, a physical frame comprises 2 bytes preamble, 2 bytes start subframe delimiter, 38 bytes PSDU and 3 bytes pause.

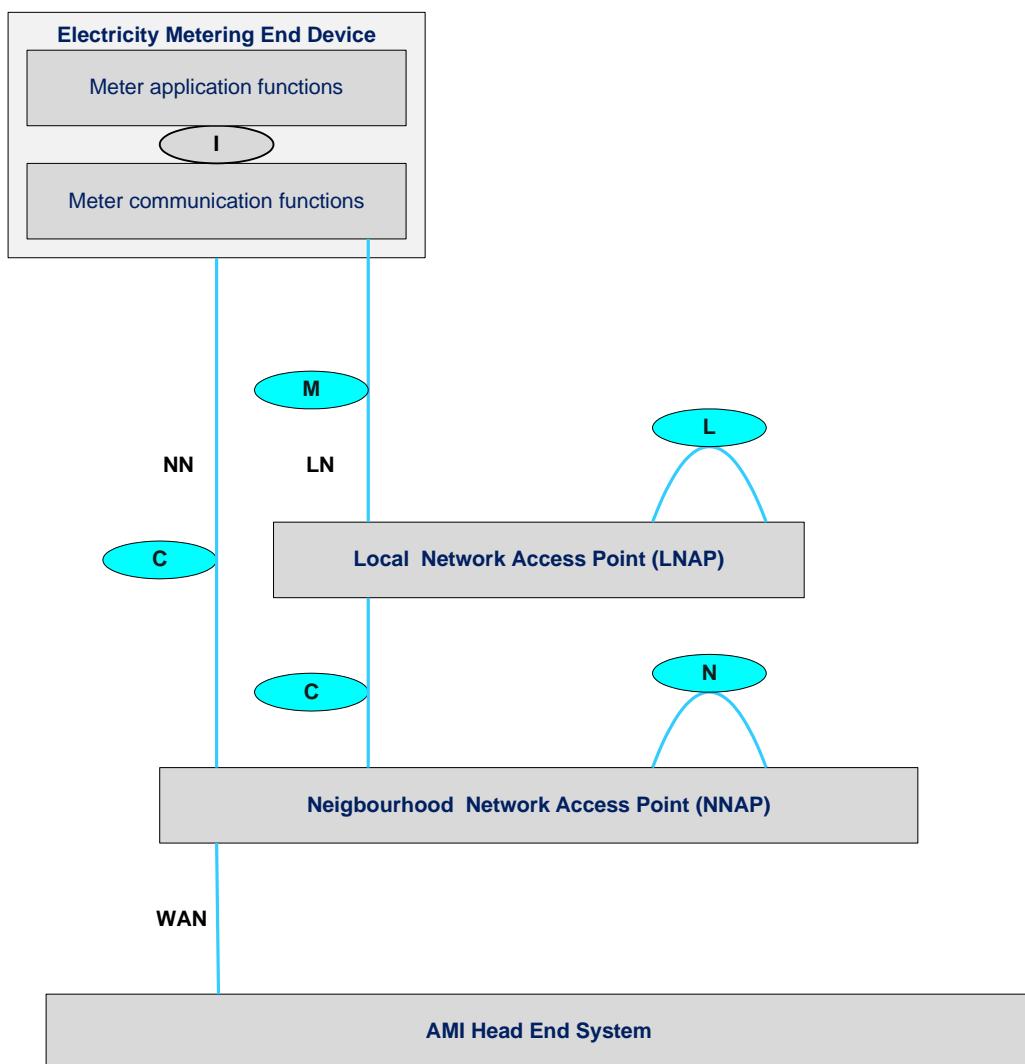
### 10.5.2 Abbreviations relevant for the S-FSK PLC profile

Abbreviation	Explanation
CIASE	Configuration Initiation Application Service Element
CI-PDU	CIASE PDU
DA	Destination Address
MPDU	MAC Protocol Data Unit
NS	Number of subframes (S-FSK MAC sublayer)
RDR	Reply Data on Request (used in IEC 61334-4-32)
SA	Source Address
SDN	Send Data Non-acknowledged (used in IEC 61334-4-32)

Abbreviation	Explanation
SMAE	Systems Management Application Entity
SMAP	Systems Management Application Process

### 10.5.3 Targeted communication environments

The *DLMS/COSEM PLC S-FSK communication profile* is intended for remote data exchange on Neighbourhood Networks (NN) between *Neighbourhood Network Access Points* (NNAP) and *Local Network Access Points* (LNAPs) or *End Devices* using S-FSK power line carrier technology over the low voltage electricity distribution network as a communication medium. The functional reference architecture is shown in Figure 166.



**Figure 166 – Communication architecture**

End devices – typically electricity meters – comprise application functions and communication functions. They may be connected directly to the NNAP via the C interface, or to an LNAP via an M interface, while the LNAP is connected to the NNAP via the C interface. The LNAP function may be co-located with the metering functions.

A NNAP comprises gateway functions and it may comprise concentrator functions. Upstream, it is connected to the Metering Head End System (HES) using suitable communication media and protocols.

End devices and LNAPs may communicate to different NNAPs, but to one NNAP only at a time. From the PLC communication point of view, the NNAP acts as an initiator while end devices and LNAPs act as responders.

NNAPs and similarly LNAPs may communicate to each other, but this is out of the Scope of this Technical Report, which covers the C interface only.

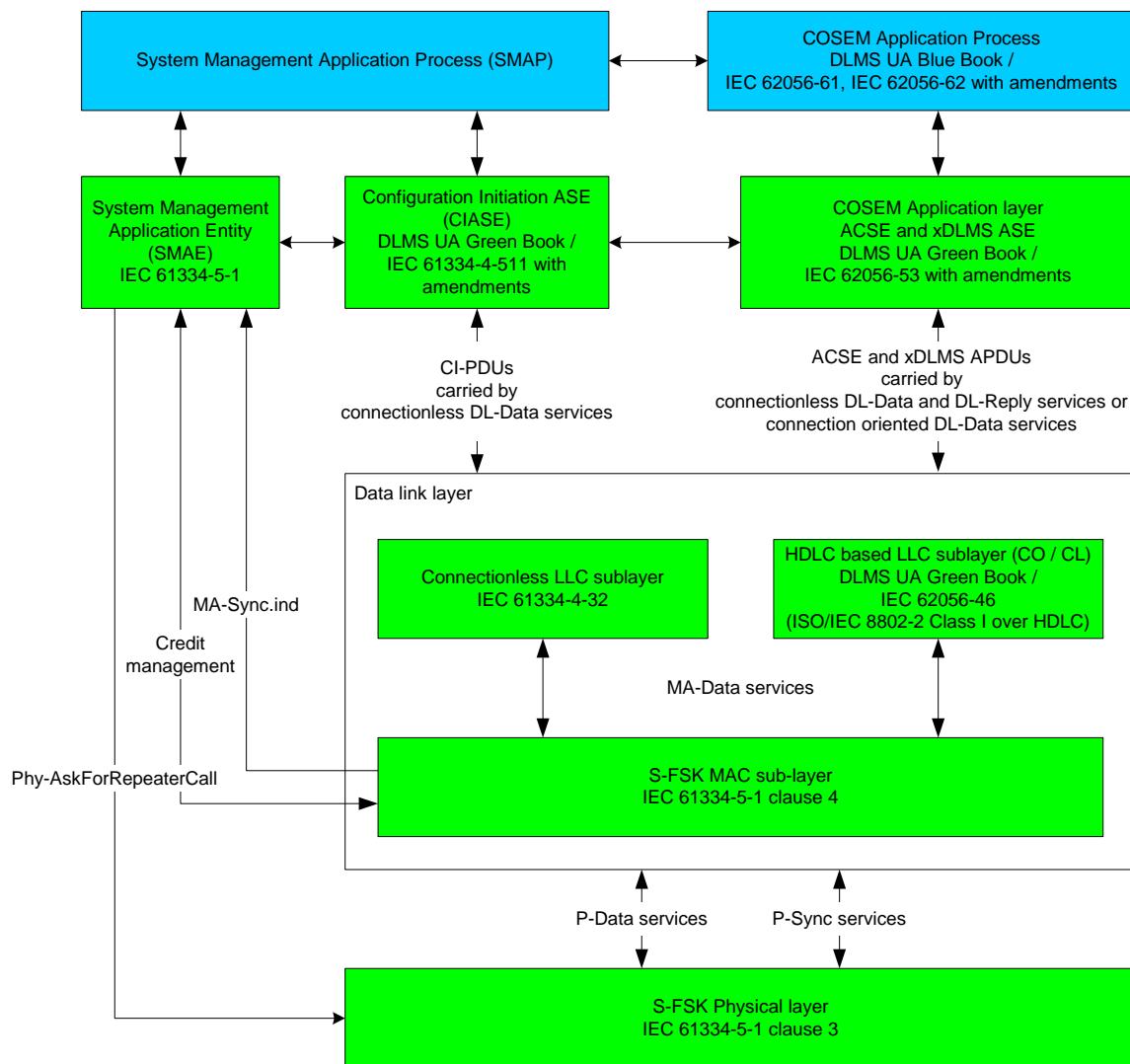
When the NNAP has concentrator functions, it acts as a COSEM client. When the NNAP has gateway functions only, then the HES acts as a DLMS client. The end devices or the LNAPs act as DLMS servers.

#### 10.5.4 Reference model

NOTE This subclause 10.5.4 is partly based on IEC 61334-4-1:1996 Clause 3.

##### 10.5.4.1 Introduction

The reference model of the *DLMS/COSEM S-FSK PLC communication profile* is shown in Figure 167.



**Figure 167 – The DLMS/COSEM S-FSK PLC communication profile**

It is based on a simplified – or collapsed – three-layer OSI architecture. The layers are the physical layer, the data link layer and the application layer. The data link layer is split to the MAC sublayer and the LLC sublayer.

#### 10.5.4.2 The physical layer (PhL)

The PhL provides the interface between the equipment and the physical transmission medium that is the distribution network. It transports binary information from the source to the destination.

The PhL in this profile is as specified in IEC 61334-5-1:2001 Clause 3. It provides the following services to its service user MAC sublayer:

- P-Data services to transfer MPDUs to (a) peer MAC sublayer entity(ies) using the LV distribution network as the transport medium;
- P-Sync services to allow the MAC sublayer entity to ask for a new synchronization and to be informed of a change in the synchronization state of the PL. These services are used locally by the MAC sublayer.

See IEC 61334-5-1:2001, 3.4.

#### 10.5.4.3 The data link layer

##### 10.5.4.3.1 General

The data link layer consists of two sublayers: the Medium Access Control (MAC) and the Logical Link Control (LLC) sublayer.

The *MAC sublayer* handles access to the physical medium and provides physical device addressing. The decision to access the medium is made by the initiator directly for its own MAC sublayer or indirectly for other MAC sublayers that are requested to transmit a response to a request sent previously by the initiator.

The *LLC sublayer* controls the logical links.

There are two LLC sublayer alternatives available:

- the connectionless LLC sublayer, as specified in IEC 61334-4-32:1996;
- the LLC sublayer using the HDLC based data link layer, as specified in Clause 8.

##### 10.5.4.3.2 The MAC sublayer

The MAC sublayer of the DLMS/COSEM S-FSK PLC communication profile is as specified in IEC 61334-5-1:2001 Clause 4. It provides the following services to its service user LLC sublayer:

- the MA-Data services. These services allow the LLC sublayer entity to exchange LLC data units with peer LLC sublayer entities. See IEC 61334-5-1:2001, 4.1.3.1;
- the MA-Sync.indication service. This allows the SMAE entity to be informed of the synchronization and configuration status of the device. See IEC 61334-5-1:2001, 4.1.3.2.

##### 10.5.4.3.3 The connectionless LLC sublayer

The connectionless LLC sublayer is as specified in IEC 61334-4-32:1996. It is derived from ISO/IEC 8802-2 – similar to Class III operation – and it performs the following functions:

- addressing of application entities within the equipment;
- sending data with no acknowledgement (SDN);
- requesting data with reply (RDR).

It provides the following services:

- DL-Data services for transporting CI-PDUs, ACSE APDUs and client-server type xDLMS APDUs;
- DL-Reply services for asking the remote LLC sublayer entity to send a previously prepared LSDU;
- DL-Update-Reply services to prepare the LSDUs to be transferred using the DL-Reply services.

For more, see IEC 61334-4-32:1996, 2.1.

##### 10.5.4.3.4 The HDLC based LLC sublayer

The HDLC based LLC sublayer is as specified in 8.

432/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

As explained in 8.1.2, this sublayer can also be divided to two sublayers:

- the LLC sublayer based on ISO/IEC 8802-2. Here, it is used in an extended Class I operation. The only role of this sublayer is to select the DLMS/COSEM application layer by using a specific LLC address. The LLC services are provided by the HDLC based MAC sublayer;
- the MAC sublayer, based on the HDLC protocol. It provides addressing of application entities within the equipment.

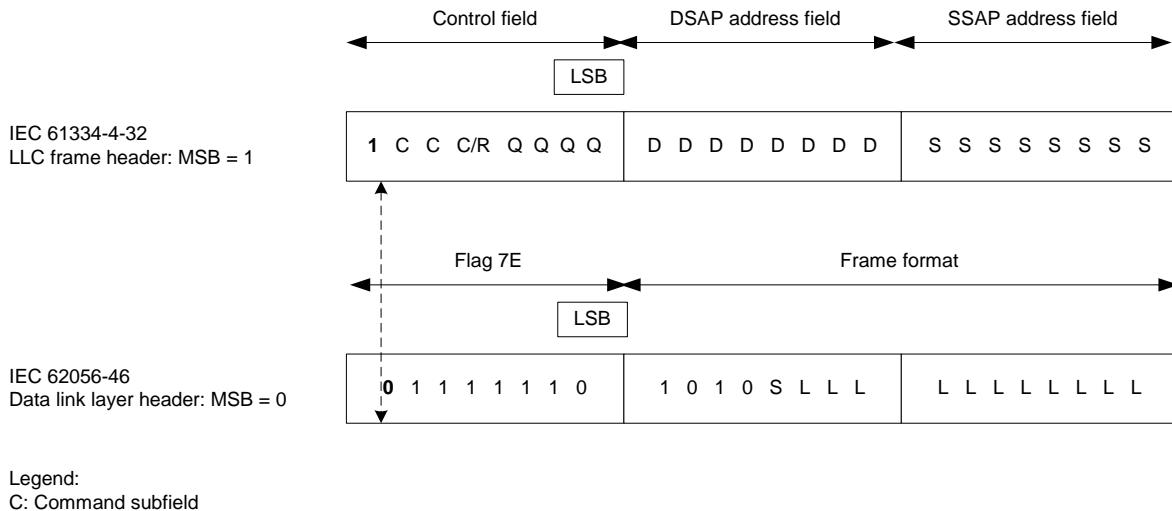
**NOTE** In this profile, there are two MAC sublayers. The HDLC MAC sublayer provides reliable LLC data transport and segmentation. The Medium Access Control functionality is provided by the S-FSK MAC sublayer specified in 10.5.4.3.2.

The HDLC based LLC sublayer provides the following services:

- DL-CONNECT services to connect and to disconnect the data link layer;
- connectionless DL-Data services for transporting CI-PDUs, ACSE APDUs and xDLMS APDUs;
- connection oriented DL-Data services for transporting ACSE APDUs and xDLMS APDUs. These services provide reliable data transport and support segmentation to carry long messages, in a transparent manner for the application layer.

#### 10.5.4.3.5 Co-existence of the connectionless and the HDLC based LLC sublayers

The frames of the connectionless LLC sublayer and the HDLC based LLC sublayer can be distinguished from each other as shown in Figure 168. This allows systems using the two profiles to co-exist on the same network.



LLC frame headers\_GK090612.wmf

**Figure 168 – Co-existence of the connectionless and the HDLC based LLC sublayers**

#### 10.5.4.4 The application layer (AL)

The application layer is the DLMS/COSEM application layer as specified in 9. It provides services to the COSEM application process (AP) and uses the services of the connectionless or the HDLC based LLC sublayer.

#### 10.5.4.5 The application process (AP)

On the server side, the COSEM device and object model – as specified in DLMS UA 1000-1 – applies. Each logical device represents an AP.

The client side APs make use of the resources of the server side AP. A physical device may host one or more client APs.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	433/614
-----------------------	------------	-----------------------	---------

### 10.5.5 The Configuration Initiation Application Service Element (CIASE)

NOTE This subclause is based on IEC 61334-4-511:2000 and constitutes an extension to it.

One of the activities of systems management is open system initialisation and / or modification. This is provided by the Configuration Initiation ASE (CIASE). It is specified in IEC 61334-4-511 with the extensions specified below.

#### 10.5.5.1 Overview

The CIASE services are the following:

- the Discover service;
- the Register service;
- the PING service;
- the RepeaterCall service; and
- the ClearAlarm service.

The three latter services, together with the Intelligent Search Initiator process specified in 10.5.5.7, constitute upper compatible functional extensions to IEC 61334-4-511:2000.

The CIASE uses the connectionless DL-Data services of the LLC sublayer.

#### 10.5.5.2 The Discover service

NOTE In this Technical Report, the description of the CIASE services follows the presentation style used for the COSEM services.

The Discover service is used to discover new systems or systems, which are in alarm state. It is specified in IEC 61334-4-511:2000, 7.1. The Discover service primitives shall provide the parameters as shown in Table 101.

**Table 101 – Service parameters of the Discover service primitives**

	Discover		DiscoverReport	
	.request	.indication	.response	.confirm
Argument	M	M (=)	–	–
Response_Probability	M	M (=)	–	–
Allowed_Time_Slots	M	M (=)	–	–
DiscoverReport_Initial_Credit	M	M (=)	–	–
IC_Equal_Credit	M	M (=)	–	–
Result (+) System_Title {System_Title} Alarm_Descriptor	–	–	S M C	S (=) M (=) C (=)
Result (-) Argument_Error(s)	–	–	S M	S (=) M (=)
NOTE This table is included here for completeness and to correct some editorial errors in IEC 61334-4-511:2000, 7.1. For the description of the service parameters, see the subclause referenced here.				

#### 10.5.5.3 The Register service

The Register service is used to perform system configuration. It assigns a MAC address to a new system identified by its system title. It is specified in IEC 61334-4-511:2000, 7.2. The Register service primitives shall provide the parameters as shown in Table 102.

**Table 102 – Service parameters of the Register service primitives**

	.request	.indication
Argument		
Active_Initiator_System_Title	M	M (=)
List_Of_Correspondence	M	M (=)
New_System_Title	M	M (=)
MAC_Address	M	M (=)
Result (+)	S	S
Result (-)	S	S
Argument_Error(s)	M	M (=)
NOTE This table is included here for completeness. For the description of the service parameters, see IEC 61334-4-511:2000, 7.2.		

NOTE 1 If a server in NEW state receives a correct Register service with its own server system title in it, it will be registered, even if it did not receive a Discover service before.

NOTE 2 Only those servers in the NEW state can be registered.

#### 10.5.5.4 The Ping Service

##### *Function*

The Ping service is used to check that a server system already registered is still present on the network. It also allows verifying that the right physical device is linked to the right MAC address. It also allows preventing the *time\_out\_not\_addressed* timer to expire.

The process begins with a Ping.request service primitive issued by the active initiator. The service contains the system title of the physical device pinged. The pingRequest CI-PDU is carried by a DL-Data.request service primitive and it is sent to the MAC address assigned to this system and to the server CIASE L-SAP.

If the system title carried by the Ping.indication service primitive is equal to the system title of the server, the server shall respond with a Ping.response service primitive, carrying the system title of the server. It is sent to the initiator CIASE L-SAP.

##### *Semantics of the service primitives*

The PING service primitives shall provide parameters as shown in Table 103.

**Table 103 – Service parameters of the PING service primitives**

	.request	.indication	.response	.confirm
Argument System_Title_Server	M	M (=)	-	-
Result (+) System_Title_Server	-	-	S M	S (=) M (=)
Result (-) Argument_Error(s)	-	-	S M	S M (=)

The System\_Title\_Server service parameter allows identifying a physical device concerned by the Ping service. The destination MAC address in the DL-Data.request service primitive is equal to the MAC address that has been assigned to this system using the Register service.

The Ping.response service primitive returns with « Result (+) » if the Ping.request service has succeeded, i.e. the system title of the physical device at the given MAC address is equal to the "System\_Title\_Server" carried by the Ping.request service primitive.

Otherwise, no response is sent by the server.

#### *Use – Client side*

The Ping.request service primitive is issued by the active initiator.

If the “System\_Title\_Server” service parameter is not valid, a local confirmation is sent immediately with a negative result indicating the problem encountered (Ping-system-title-nok).

Otherwise, the CIASE forms a DL-Data.request PDU containing a pingRequest CI-PDU that carries the System\_Title\_Server requested. It is sent to the physical device concerned by the request.

Once the transmission of the pingRequest CI-PDU is over, the CIASE waits for a DL-Data.indication service primitive containing a pingResponse CI-PDU from the physical device pinged, during the necessary time that depends on the initial credit of the request.

If the CIASE receives a DL-Data.indication service primitive containing a pingResponse CI-PDU before this delay is over, it sends to the initiator a confirmation with a positive result, containing the service parameter returned by the server system.

If no DL-DATA.indication service primitive is received, the CIASE sends to the initiator a confirmation with a negative result pointing out the absence of an answer (Ping-no-response).

#### *Use – Server side*

On the reception of a DL-Data.indication service primitive containing a pingRequest CI-PDU, the CIASE checks that the System\_Title\_Server service parameter is correct and that it is equal to its own system title.

If so, it invokes a Ping.response service primitive that includes its system title. The pingResponse CI-PDU is carried by a DL-Data.request service primitive.

If the service parameter of the Ping.indication service primitive is not correct, no response is sent.

Finally, if the System\_Title\_Server service parameter in the Ping.indication service primitive is correct, but not equal to the system title of the physical device, no response is sent.

#### **10.5.5.5 The RepeaterCall service**

##### *Function*

The purpose of the RepeaterCall service is to adapt the repeater status of server systems depending on the topology of the electrical network. It allows the automatic configuration of the repeater status on the whole network.

In the RepeaterCall mode, the client and the servers transmit short frames – two bytes long each – and measure the level of the signal to determine if a server system should be a repeater or not.

##### *Semantics of the service primitives*

The RepeaterCall service primitives shall provide parameters as shown in Table 104.

**Table 104 – Service parameters of the RepeaterCall service primitives**

	. request	. indication
Arguments		
Max_Ad MAC	M	M (=)
Nb_Tslot_For_New	U	U (=)
Reception_Threshold	M	M (=)
Result (+)	S	S
Result (-)	S	S
Argument_Error(s)	M	M

The Max\_Ad MAC service parameter allows calculating the number of timeslots used in the RepeaterCall mode by the physical layer of the server systems registered to an initiator. It corresponds to the largest server system MAC address that is stored by the initiator.

NOTE 1 The largest allowable server MAC addresses is 3071 (BFF), as the range C00...DFF is reserved for initiator, as specified in IEC 61334-5-1 4.3.7.7.1.

The value of Nb\_Tslot is calculated from this information:

$$Nb\_Tslot = [MaxAdrMax/21]+1$$

where  $[x]$  means the floor of  $x$ , the nearest integer  $\leq x$ .

Example 1:

Max\_Ad MAC = 20 (20 servers on the network)

$$Nb\_Tslot = [20/21]+1 = 1$$

Nb\_Tslot = 1, with 1 sub-timeslot for the concentrator and 20 sub-timeslots for the servers.

Example 2:

Max\_Ad MAC = 21 (21 servers on the network)

$$Nb\_Tslot = [21/21]+1=2$$

Nb\_Tslot = 2:

- 1 timeslot with 1 sub-timeslot for the concentrator and 20 sub-timeslots for 20 servers;
- 1 timeslot with 1 sub-timeslot for one server.

The Nb\_Tslot\_For\_New service parameter defines the number of timeslots used in the RepeaterCall mode by the physical layer of the server systems in NEW state. If the value of this service parameter is 0 (or not present) then the systems in NEW state are not allowed to participate in the Repeater Call process.

The maximum number of timeslots used by the physical layer is equal to the sum of the number of timeslots for the server systems registered and the number of timeslots for the server systems in NEW state.

The Reception\_Threshold service parameter defines the threshold of the signal level in dB $\mu$ V, necessary to validate a physical pattern in a Sub\_Tslot when the physical layer is in the RepeaterCall mode.

The Result (+) service parameter (positive result) indicates that the requested service has succeeded.

The Result (-) service parameter (negative result) indicates that the requested service has failed.

The “Arguments Error” indicates that at least one argument has a wrong value.

### *Use – Client side*

The RepeaterCall service of the CIASE is invoked by the SMAP.

If any of the arguments is not valid, a confirmation is sent immediately with a negative result indicating the problem encountered.

Otherwise, the CIASE forms a DL-Data.request service primitive containing a repeaterCall CI-PDU carrying the parameters requested. This request is sent to all server systems.

A positive confirmation is passed to the CIASE upon the reception of a DL-Data.cnf(+).

When this confirmation is received, the CIASE sends the Phy\_AskForRepeaterCall.request primitive allowing the activation of the RepeaterCall mode of the physical layer. The parameters of this primitive are the following:

- Sub\_Tslot position: on the client (initiator) side its value is 0;
- Reception\_Threshold.

NOTE 2 On the client side, the Reception\_Threshold parameter has no significance.

### *Use – server side*

On the server side, on the reception of a DL-Data.indication service primitive containing a repeaterCall CI-PDU, the CIASE verifies that the service parameters are correct.

If this is the case, it sends the Phy\_AskForRepeaterCall.request primitive to activate the RepeaterCall mode of the physical layer. The parameters of this primitive are the following:

- Sub\_Tslot position: A number expressed on two octets, between 0 and 65 535. The value 0 is reserved for the configuration of the client (concentrator). The other values are available for the configuration of server systems;

In the case of server systems registered by a concentrator, Sub\_Tslot takes the value of the local MAC address of the server system, between 1 and Max\_Ad MAC. For the server systems not registered, Sub\_Tslot takes a random value between Max\_Ad MAC and Max\_Ad MAC + (Nb\_Tslot\_For\_New \* 21).

NOTE 3 Max\_Ad MAC and Nb\_Tslot\_For\_New are the service parameters of the RepeaterCall.request service.

- Reception\_Threshold: This represents the signal level in dB $\mu$ V. The default value is 104.

If the response to this request is negative, the command is cancelled. The following cases lead to a failure:

- the state of the physical layer is not correct (there is no physical synchronization);
- the repeater status is never\_repeater;
- the parameters are incorrect.

The participation of the servers in the repeater call process and the effect of the process on their repeater status depend on the *repeater* management variable (attribute 10 of the S-FSK Phy&MAC setup object) and the signal level heard:

- servers configured as never repeater do not participate: they do not transmit during their sub-timeslot and their repeater\_status (attribute 11 of the S-FSK Phy&MAC setup object) is not affected;
- servers configured as always repeater participate: they transmit during their timeslot but their repeater\_status is not affected;
- servers configured as dynamic repeater participate: they transmit during their sub-timeslot, if they have not heard a signal before from the client or from any servers, which is above the reception threshold. If during the whole repeater call process, a server does not hear a signal from the client or from other servers, which is above the reception threshold, then its repeater status will be TRUE: the server will repeat all frames. If a server hears a signal from the client or from other servers, which is above the reception threshold, then its repeater status will be FALSE: the server won't repeat any frames.

**NOTE 4** If each server configured as dynamic repeater hears a signal which is above the reception threshold, this means that they are all close to a client and no repetition is needed. So, none of them will become a repeater.

#### 10.5.5.6 The ClearAlarm service

##### Function

The ClearAlarm service allows clearing the alarm state in (a) server system(s), in a point-to-point or in a broadcast mode.

##### Semantics of the service primitive

The ClearAlarm service primitives shall provide parameters as shown in Table 105.

**Table 105 – Service parameters of the ClearAlarm service primitives**

	.request	.indication
Arguments		
Alarm_Descriptor	S	S (=)
Alarm_Descriptor {Alarm_Descriptor}	S	S (=)
Alarm_Descriptor_List_And_Server_List	S	S (=)
System_Title {System_Title}	M	M (=)
Alarm_Descriptor {Alarm_Descriptor}	M	M (=)
Alarm_Descriptor_By_Server {Alarm_Descriptor_By_Server}	S	S (=)
System_Title	M	M (=)
Alarm_Descriptor	M	M (=)
Result (+)	S	S
Result (-)	S	S
Arguments Error	M	M

This service provides four different possibilities:

- the Alarm\_Descriptor choice allows clearing a single alarm in all server systems. The value of the Alarm\_Descriptor parameter identifies the alarm to be cleared;
- the Alarm\_Descriptor {Alarm\_Descriptor} choice allows clearing a list of alarms in all server systems. The value of the Alarm\_Descriptor {Alarm\_Descriptor} parameter identifies the list of alarms to be cleared;
- the Alarm\_Descriptor\_List\_And\_Server\_List choice allows clearing a common list of alarms specified in the list of server systems specified. The System\_Title {System\_Title} parameter identifies the list of server systems in which the alarms have to be cleared. The value of the Alarm\_Descriptor {Alarm\_Descriptor} parameter identifies the list of alarms to be cleared;

- the `Alarm_Descriptor_By_Server` {`Alarm_Descriptor_By_Server`} choice allows clearing one alarm specified in each server specified. The `System_Title` parameter identifies the server system in which the alarm has to be cleared. The value of the `Alarm_Descriptor` parameter identifies the alarm to be cleared.

NOTE As specified in IEC 61334-4-511:2000, 6.2.2, alarm descriptors should be specified in Technical Reports.

The Result (+) argument (positive result) indicates that the requested service has succeeded.

The Result (-) argument (negative result) indicates that the requested service has failed.

The "Argument Error(s)" indicates that at least one argument has a wrong value.

#### Use

The ClearAlarm CIASE service is invoked by the initiator.

If any of the service parameters is not valid, a confirmation is sent immediately with a negative result indicating the problem encountered.

Otherwise, the CIASE forms a DL-Data.request service primitive containing a clearAlarm CI-PDU containing the parameters requested. This request is sent to the server system(s) concerned by the request. A positive confirmation is sent upon the reception of a DL-Data.cnf(+) service primitive.

On the server side, on the reception of a DL-Data.indication service primitive containing a clearAlarm CI-PDU, the CIASE verifies that the arguments are correct. If this is the case, it clears the alarms corresponding to the list of alarms. Otherwise, the service is ignored.

### 10.5.5.7 The Intelligent Search Initiator process

#### 10.5.5.7.1 Introduction

The objective of the Intelligent Search Initiator process is to improve plug&play installation of server systems, by ensuring that each server system is registered by the correct initiator.

When a new server system is placed on the network, it will be discovered and registered by the first initiator it hears talking. It remains registered by that initiator as long as it keeps receiving correct frames (the `time_out_not_addressed` timer does not expire). If there is cross-talk on the network, the server system may be registered by the wrong initiator, i.e. one, which is "heard" by the server system due to cross-talk.

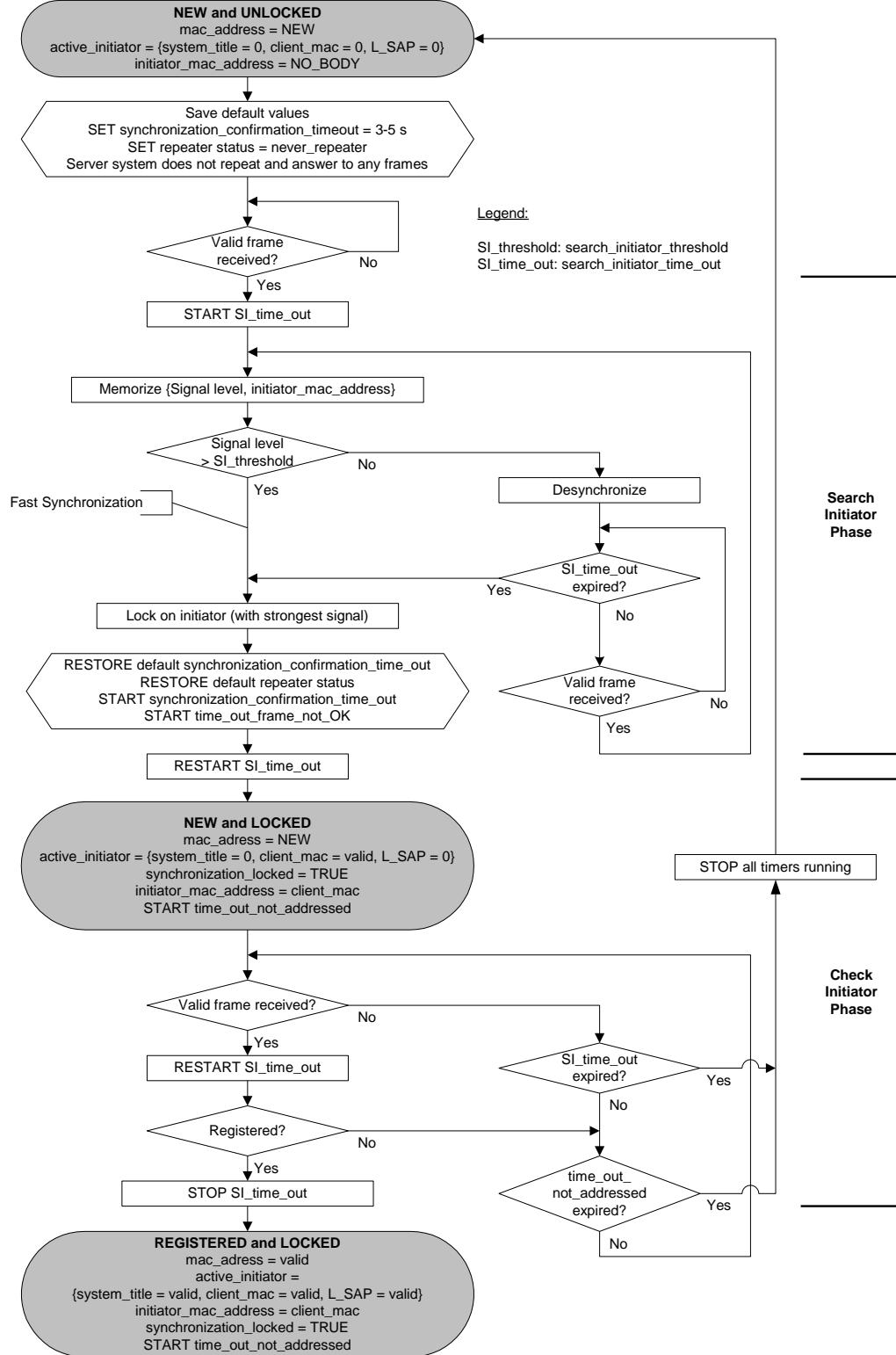
When the Intelligent Search Initiator process is implemented in the server system, it is capable to establish a list of all initiators it can "hear", and to lock on the initiator with the best signal level.

#### 10.5.5.7.2 Operation

##### 10.5.5.7.2.1 Flow chart

The intelligent search initiator process comprises two phases:

- the *Search Initiator* phase;
- the *Check Initiator* phase.



**NOTE** A valid frame is a frame in which either the source or the destination address is an initiator address, and otherwise correct.

**Figure 169 – Intelligent Search Initiator process flow chart**

The Intelligent Search Initiator process is shown in Figure 169. See also Figure 170 showing the complete discovery and registration process.

#### 10.5.5.7.2.2 Process parameters

The Intelligent Search Initiator process is characterized by two parameters:

- The search\_initiator\_time\_out, defining the duration of the Search Initiator Phase. This timeout is also used during the Check Initiator Phase, see 10.5.5.7.2.4;  
The Search Initiator Phase must be long enough to allow the server systems to hear all the initiators around them and, when needed, to let sufficient time to start all the initiators by the central station. However, this time must not be too long either, so that the discovery phase could be executed correctly. The recommended value for this timeout is 10 minutes;
- The search\_initiator\_threshold, defining the minimum signal level allowing a fast synchronization.

The value of the search\_initiator\_threshold must be chosen so that the server systems next to an initiator lock on it immediately, but the server systems a bit further away (maybe on an other network) do only lock on it after the search\_initiator\_time\_out timer expires. The default value is 98 dB<sub>μ</sub>V.

#### 10.5.5.7.2.3 Search Initiator Phase

Initially, the server system is in the NEW and UNLOCKED state waiting to receive a valid frame.

For the Search Initiator Phase, the synchronization\_confirmation\_time\_out shall be reduced to 3-5 s. Otherwise, a server system would remain synchronized too long on a bad frame.

During this phase, a server system must never repeat any frames. This is because if repetition was allowed, it would have to repeat all frames, not only the frames from the closest initiator, and so the other server systems next to it would listen to frames from a bad initiator with a strong signal level. Therefore, the Intelligent Search Initiator algorithm can only be efficient if all the server systems on a network have the algorithm implemented (otherwise, other server systems could repeat frames and foul the signal level).

For the same reasons, a server system must never transmit any frames during the Search Initiator Phase:

- it should not answer to a Discover request (It should be desynchronized before, except if the signal level is good enough to allow fast synchronization. But in this case, the server system is not in the Search Initiator Phase anymore but in the Check Initiator Phase.);
- it should not answer to a Register request either;
- and in particular, it should not answer any ACSE or xDLMS service requests (this is obvious because the server system is in the NEW state).

Notice that as soon as a server system becomes locked, it can repeat frames since it will only accept frames from the good initiator. (It is even advised that it repeats frames, since it will shorten the Search Initiator Phase for the other server systems).

Each time that the server system receives a frame, it checks the signal level and the MAC addresses in it:

- if none of the MAC addresses – source or destination – is an initiator MAC address, the frame is considered as invalid; the server system immediately desynchronizes in order to listen to another frame;
- if one of the MAC addresses is an initiator MAC address, and the signal level is good enough (signal level > search\_initiator\_threshold – see 10.5.5.7.2.2) the server system locks on that initiator. This is called Fast Synchronization. This occurs when the server system is next to an initiator or to a server system that is already registered to that initiator. The server system enters the Check Initiator Phase;
- if one of the MAC addresses is an initiator MAC address but the signal level is not good enough (signal level < search\_initiator\_threshold), the server system memorizes the signal level and the MAC address, then desynchronizes immediately in order to listen to another frame;
- when the search\_initiator\_time\_out expires, the server system locks to the initiator having provided the best signal level.

At this point:

- the default values of the synchronization\_confirmation\_time\_out and the repeater status are restored;
- the synchronization\_confirmation\_time\_out and the time\_out\_frame\_not\_OK timers are initialised;
- the search\_initiator\_time\_out is restarted.

The server is in the NEW and LOCKED state and enters the Check Initiator Phase.

#### 10.5.5.7.2.4 Check Initiator Phase

Once the server system is locked, it can be discovered and registered. The process is the following:

- the time\_out\_not\_addressed timer is started;
- the server waits then to be discovered and registered;
- the search\_initiator\_time\_out timer is restarted each time a valid frame is received;
- when the server system is registered (valid frame carrying a register CI-PDU received):
  - the search\_initiator\_time\_out timer is stopped;
  - the active\_initiator variable is set;
- if either the search\_initiator\_time\_out or the time\_out\_not\_addressed timer expires, it means that the server system did not receive a frame from or to its initiator for a long time: all timers are stopped then and the server system returns to the initial state: NEW and UNLOCKED.

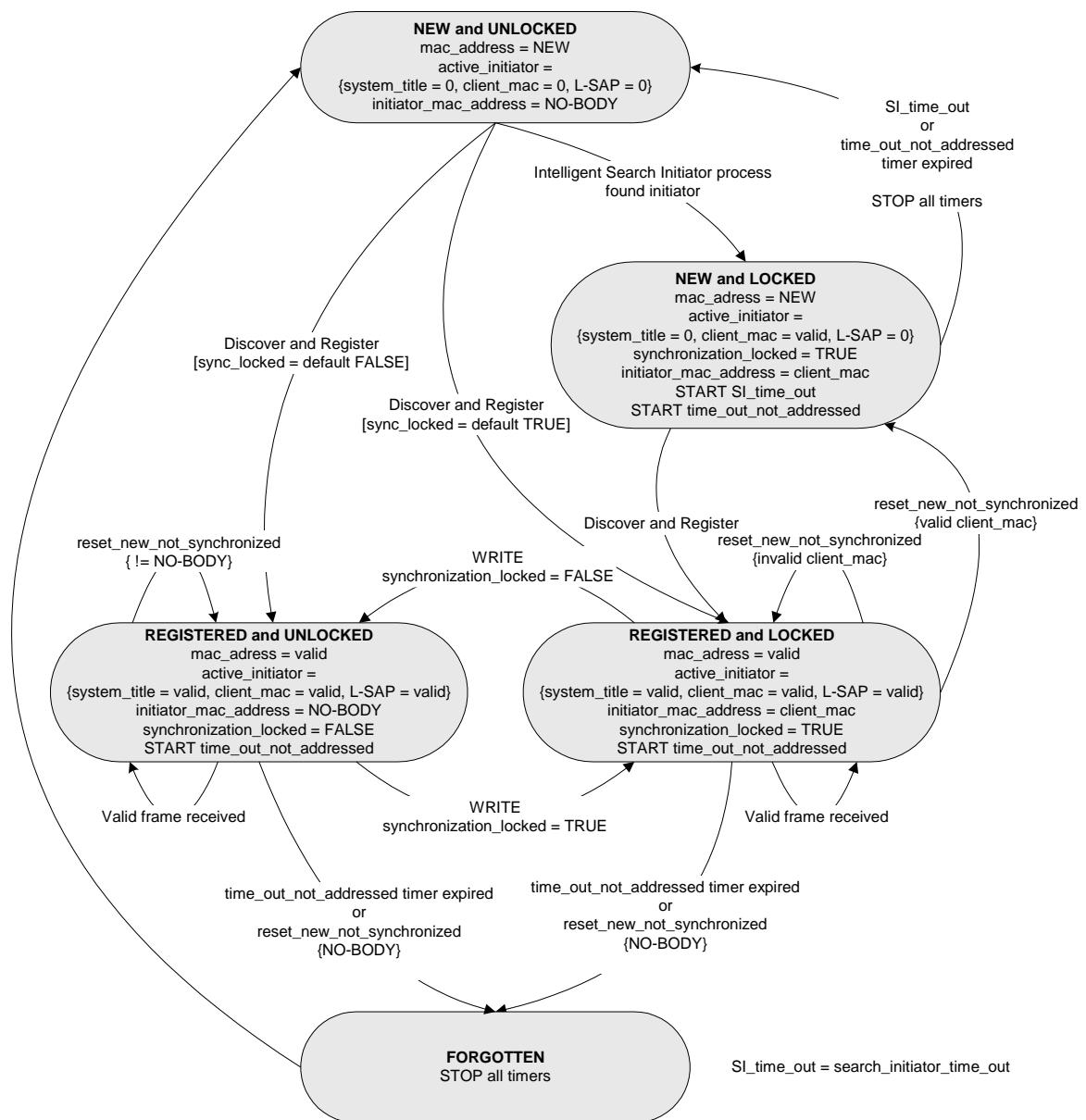
#### 10.5.5.7.2.5 Remarks

The Intelligent Search Initiator process has to be used cautiously. If a server system hears a frame with a wrong MAC address, it will lock on it and won't unlock until the search\_initiator\_time\_out is over. It is advised not to change the initiator MAC address of a concentrator unless it can be made sure that all the server systems on the network are in the unconfigured state: NEW and UNLOCKED.

#### 10.5.5.8 The Discovery and Registration process

The Discovery and Registration process, including the Intelligent Search Initiator process, is summarized in Figure 170.

NOTE 1 The state transitions caused by writing the mac-address and initiator-mac-address variables are not shown.

**Figure 170 – The Discovery and Registration process**

The process is the following.

Initially, the server is in the NEW and UNLOCKED (UNCONFIGURED) state. In this state:

- mac\_address = NEW;
- active\_initiator: default value, with all three elements set to 0:
  - system\_title = octet string of 0s;
  - (client\_)mac\_address = NO-BODY;
  - L-SAP\_selector = 0;
- initiator\_mac\_address = NO-BODY.

The default value of the synchronization\_locked variable is as specified in the Technical Report:

1. FALSE: the value of initiator\_mac\_address variable is always NO-BODY;
2. TRUE: the value of the initiator\_mac\_address variable follows the value of the (client\_)mac\_address element of the active\_initiator variable;
3. The Intelligent Search Initiator process is used. In this case, the value of the search\_initiator\_time\_out variable shall be different from 0. The server moves from the NEW and UNLOCKED state to the NEW and LOCKED state at the end of the Search Initiator Phase.

NOTE 2 This third possibility is an extension to IEC 61334-4-511:2000.

When a server is installed, it has to be discovered and registered.

In case 1), upon the reception of a valid Register CIASE PDU the server moves to the REGISTERED and UNLOCKED state:

- the mac\_address variable is set to the value allocated by the initiator;
- the elements of the active\_initiator variable are set to the values contained in the register CIASE PDU and the DL-Data.indication LLC PDU:
  - system\_title: system title of the initiator;
  - (client\_)MAC\_address: MAC address of the initiator;
  - L-SAP\_selector: the L-SAP used by the initiator;
- the initiator\_MAC\_address remains at NO-BODY;
- the synchronization\_locked variable remains at FALSE;
- the time\_out\_not\_addressed timer is started.

In case 2), upon the reception of a valid Register CIASE PDU, the server moves to the REGISTERED and LOCKED state:

- the mac\_address variable is set to the value allocated by the initiator;
- the elements of the active\_initiator variable are set to the values contained in the register CIASE PDU and the DL-Data.indication LLC PDU:
  - system\_title: system title of the initiator;
  - (client\_)MAC\_address: MAC address of the initiator;
  - L-SAP\_selector: the L-SAP used by the initiator;
  - the initiator\_MAC\_address is updated to be equal to the (client\_)MAC\_address element of the active\_initiator;
  - the synchronization\_locked attribute remains at TRUE;
  - the time\_out\_not\_addressed timer is started.

In case 3), the server searches first for the initiator providing the strongest signal; see Figure 169. At the end of Search Initiator Phase, the search\_initiator\_time\_out is re-started, the time\_out\_not\_addressed timer is started, and the server locks on the initiator chosen:

- the (server\_)mac\_address variable is still at NO-BODY (server is in NEW state);
- the synchronization\_locked variable is set to TRUE;
- the (client\_)MAC\_address element of the active\_initiator variable takes the MAC\_address of the initiator chosen. The other elements of the active\_initiator variable remain at their default value;
- the initiator\_MAC\_address is updated to be equal to the (client\_)MAC\_address element of the active\_initiator variable.

The server is in the NEW and LOCKED state and enters the Check Initiator Phase.

The server can only be registered by the initiator chosen. This takes place exactly as in case 2). If the server is not registered before either the search\_initiator\_time\_out or the time\_out\_not\_addressed timer expires, it returns to the NEW and UNLOCKED state. All timers are stopped.

In the REGISTERED state, the server may receive frames addressed to it and respond to them. The time\_out\_not\_addressed timer is restarted with each frame addressed to the server.

The server may also move from the REGISTERED and UNLOCKED state to the REGISTERED and LOCKED state and vice versa by writing the value of the synchronization\_locked variable.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	445/614
-----------------------	------------	-----------------------	---------

The server may leave the REGISTERED state either:

- 1) by the expiration of the time\_out\_not\_addressed timeout; or
- 2) by writing the reset\_new\_not\_synchronized variable.

In case 1), the server becomes “FORGOTTEN”: it loses its MAC address and its active initiator: the mac\_address, the initiator\_mac\_address and the active\_initiator variables are all reset. The server returns to the NEW and UNLOCKED state.

In case 2) the server checks first the value of the client\_mac\_address submitted as a parameter of the reset\_new\_not\_synchronized request:

- if the value is not equal to a valid client address, or the pre-defined NO-BODY address, the writing is refused;
- if the value is NO-BODY, it has the same effect as the expiration of the time\_out\_not\_addressed timer: the server returns to the NEW and UNLOCKED state;
- if the value is a valid client address, then the value of the synchronization\_locked variable is checked:
  - if it is FALSE – the server is in the REGISTERED and UNLOCKED state – the writing is refused;
  - if it is TRUE – the server is in the REGISTERED and LOCKED state – the (client\_)mac\_address element of the active initiator variable is set to the value submitted, the system\_title and the L-SAP\_selector elements are reset to 0. The initiator\_mac\_address variable is also set to the value submitted. The time\_out\_not\_addressed timer is stopped. The server returns to the NEW and LOCKED state, where it waits to be registered again by the initiator it has been reset to.

When the server leaves the REGISTERED state, all AAs are aborted.

If a power failure occurs, it is managed as follows:

- if the server is in the NEW and UNLOCKED state, it will stay there when the power returns;
- if the server is in the Search Initiator Phase, then it moves back to the NEW and UNLOCKED state when the power returns. The search\_initiator\_time\_out timer is stopped and all data concerning the initiators heard so far (signal level and MAC address) are lost;
- if the server is in the NEW and LOCKED state, it will stay there when the power returns. The search\_initiator\_time\_out and the time\_out\_not\_addressed timers are restarted;
- if the server is in the REGISTERED (LOCKED or UNLOCKED) state, it will stay there when the power returns. The time\_out\_not\_addressed timer is restarted. The Application Associations open before the power failure are locally re-established.

### **10.5.5.9 Abstract and transfer syntax**

As specified in IEC 61334-4-511:2000, 7.3.1, the CIASE uses the ASN.1 abstract syntax. The transfer syntax is A-XDR, see 10.5.9.

## **10.5.6 Addressing**

### **10.5.6.1 General**

In the DLMS/COSEM S-FSK PLC profile, two levels of addresses are defined:

- at the MAC sublayer that processes MAC addresses to access an LLC entity;
- at the LLC sublayer that processes LLC addresses to access application entities.

### **10.5.6.2 IEC 61334-5-1 MAC addresses**

A physical client or server – initiator or responder – system may be accessed using a MAC address specific to the system or by using one of the group MAC addresses.

**Table 106 – MAC addresses**

Address	Value	Reference
NO-BODY	000	IEC 61334-4-1:1996, 4.3.2.6, IEC 61334-5-1:2001, 4.3.7.5.1
Local MAC	001...FIMA-1	IEC 61334-4-1:1996, 4.3.2.5
Initiator	FIMA...LIMA	IEC 61334-4-1:1996, 4.3.2.4
MAC group address	LIMA + 1...FFB	
All-configured	FFC	IEC 61334-4-1:1996, 4.3.2.1, IEC 61334-5-1:2001, 4.3.7.5.2
NEW	FFE	IEC 61334-4-1:1996, 4.3.2.2, IEC 61334-5-1:2001, 4.2.3.2
All Physical	FFF	IEC 61334-4-1:1996, 4.3.2.3, IEC 61334-5-1:2001, 4.3.7.5.3
NOTE MAC addresses are expressed on 12 bits. FIMA = First initiator MAC address; C00 LIMA = Last initiator MAC address; DFF		

### 10.5.6.3 Reserved special LLC addresses

NOTE This subclause 10.5.6.3 is based on IEC 61334-4-1:1996. 4.4.

#### 10.5.6.3.1 General

Each application process within the physical device is bound to a data link layer address that consists of the doublet {MAC-address, L-SAP}. The following LLC addresses (L-SAPs) are specified:

- All-L-SAP: designates the group consisting of all L-SAPs actively serviced by the underlying MAC layer (with its specified MAC address);
- system management L-SAP (M-L-SAP): there is only one M-L-SAP in a physical system;
- initiator L-SAP (I-L-SAP): These are defined in a specific range;
- individual LLC addresses: These are defined in a specific range;
- CIASE L-SAP (C-L-SAP).

#### 10.5.6.3.2 Reserved addresses for the IEC 61334-4-32 LLC sublayer

The reserved LLC addresses for the IEC 61334-4-32:1996 LLC sublayer on the client side and the server side are shown in Table 107 and Table 108 respectively.

**Table 107 – Reserved IEC 61334-4-32 LLC addresses on the client side**

Address	L-SAP	Meaning
0x00		No-station
0x01	M-L-SAP I-L-SAP	Client Management Process. The CIASE is also bound to this address.
0x10		Public Client (lowest security level)

**Table 108 – Reserved IEC 61334-4-32 LLC addresses on the server side**

Address	L-SAP	Meaning
0x00	I-L-SAP	CIASE
0x01	M-L-SAP	Management Logical Device
0x02...0x0F		Reserved for future use
0xFF	All-L-SAP	All-station (Broadcast)

### 10.5.6.3.3 Reserved addresses for the HDLC based LLC sublayer

The reserved LLC addresses for the HDLC based LLC sublayer on the client side and the server side are shown in Table 109 and Table 110 respectively.

**Table 109 – Reserved HDLC based LLC addresses on the client side**

Address	L-SAP	Meaning
0x00		No-station
0x01	M-L-SAP I-L-SAP	Client Management Process. The CIASE is also bound to this address.
0x10		Public Client

**Table 110 – Reserved HDLC based LLC addresses on the server side**

One byte address	Two bytes address	Meaning
0x00	0x0000	No-Station. The CIASE is also bound to this address.
0x01	0x0001	Management Logical Device
0x02...0x0F	0x0002...0x000F	Reserved for future use
0x7E	0x3FFE	Calling Physical Device address; not used in the S-FSK HDLC based profile.
0x7F	0x3FFF	All-station (Broadcast)

### 10.5.6.4 Source and destination APs and addresses of CI-PDUs

The Source and destination APs and addresses of CI-PDU requests are show in Table 111.

**Table 111 – Source and Destination APs and addresses of CI-PDUs**

CI-PDU	Source AP	Destination AP	MAC SA	MAC DA	D-L-SAP	S-L-SAP
discover	Initiator	AIISMAE	Initiator	FFF All-Physical	0x00 CIASE	0x01 CIASE <sup>1</sup>
discoverReport	Manager	AIISMAE	NEW FFE <sup>2</sup> or individual server MAC	FFF <sup>3</sup> All-Physical or Initiator	0xFD	0x00 CIASE
register	Initiator	AIISMAE	Initiator	FFF All-Physical	0x00 CIASE	0x01 CIASE <sup>4</sup>
pingRequest	Initiator	Individual	Initiator	Individual	0x00 CIASE	0x01 CIASE <sup>4</sup>
pingResponse	Individual	Initiator	Individual	Initiator	0x01 CIASE <sup>4</sup>	0x00 CIASE
repeaterCall	Initiator	AIISMAE	Initiator	FFF All-Physical	0x00 CIASE	0x01 CIASE <sup>4</sup>
clearAlarm	Initiator	AIISMAE	Initiator	FFF All-Physical	0x00 CIASE	0x01 CIASE <sup>4</sup>

<sup>1)</sup> Could be a different value  
<sup>2)</sup> FFE if the server is in the NEW state. Individual MAC address if the server is in ALARM state.  
<sup>3)</sup> If the reporting system list feature is used, then the Destination Address is All-Physical. Otherwise, it is the Initiator address.  
<sup>4)</sup> Could be different value, but must be the same as in the discover CI-PDU.

CI-PDU	Source AP	Destination AP	MAC SA	MAC DA	D-L-SAP	S-L-SAP
NOTES						
In the MAC frame, the order of the addresses is Source Address – Destination Address.						
In the IEC 61334-4-32 LLC frame, the order of the addresses is Destination Address – Source Address.						
In the HDLC frame, the order of the addresses is Destination Address – Source Address.						

### 10.5.7 Specific considerations / constraints for the IEC 61334-4-32 LLC sublayer based profile

#### 10.5.7.1 Establishing application associations

AAs can only be established with server systems properly registered by the initiator. The MSC for the discovery and registration process is shown in Figure 171.

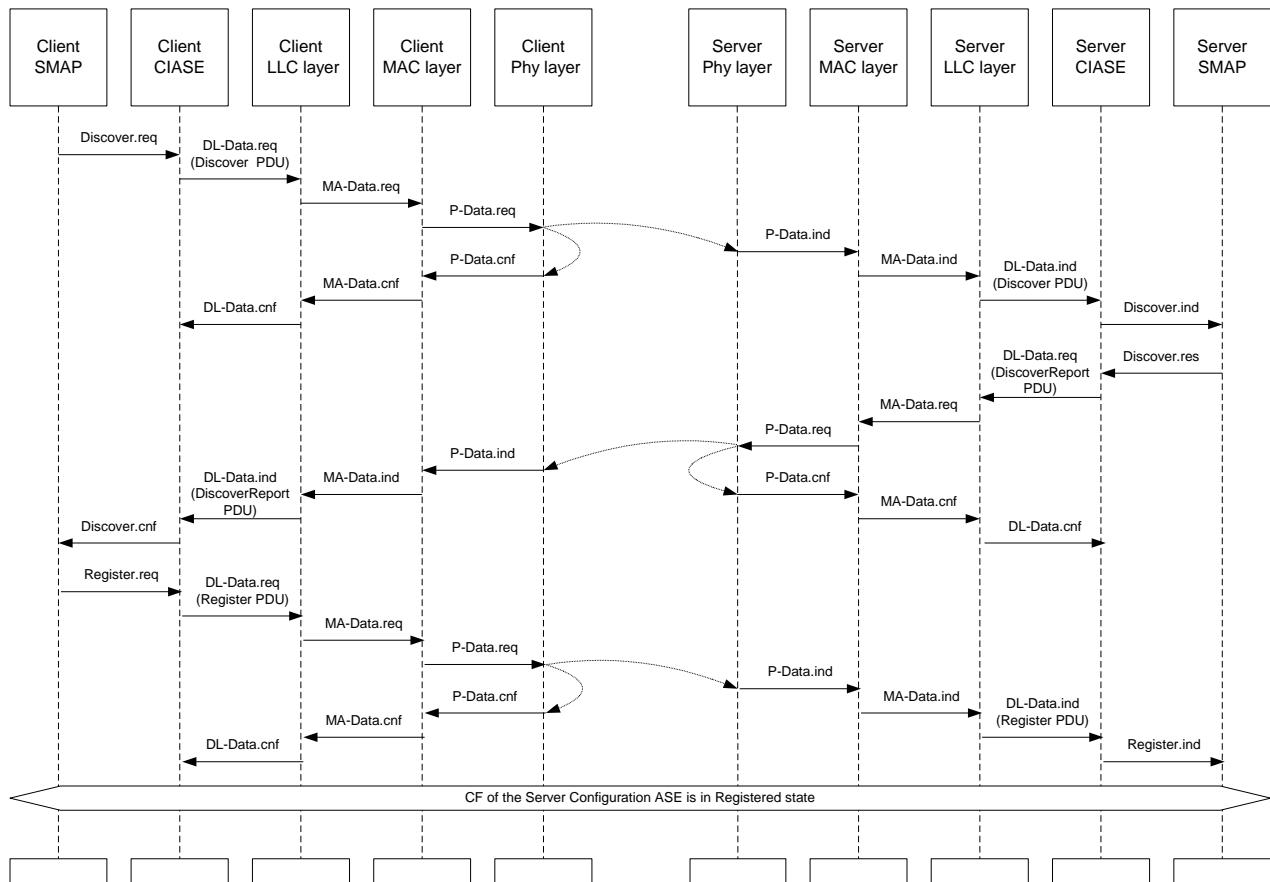
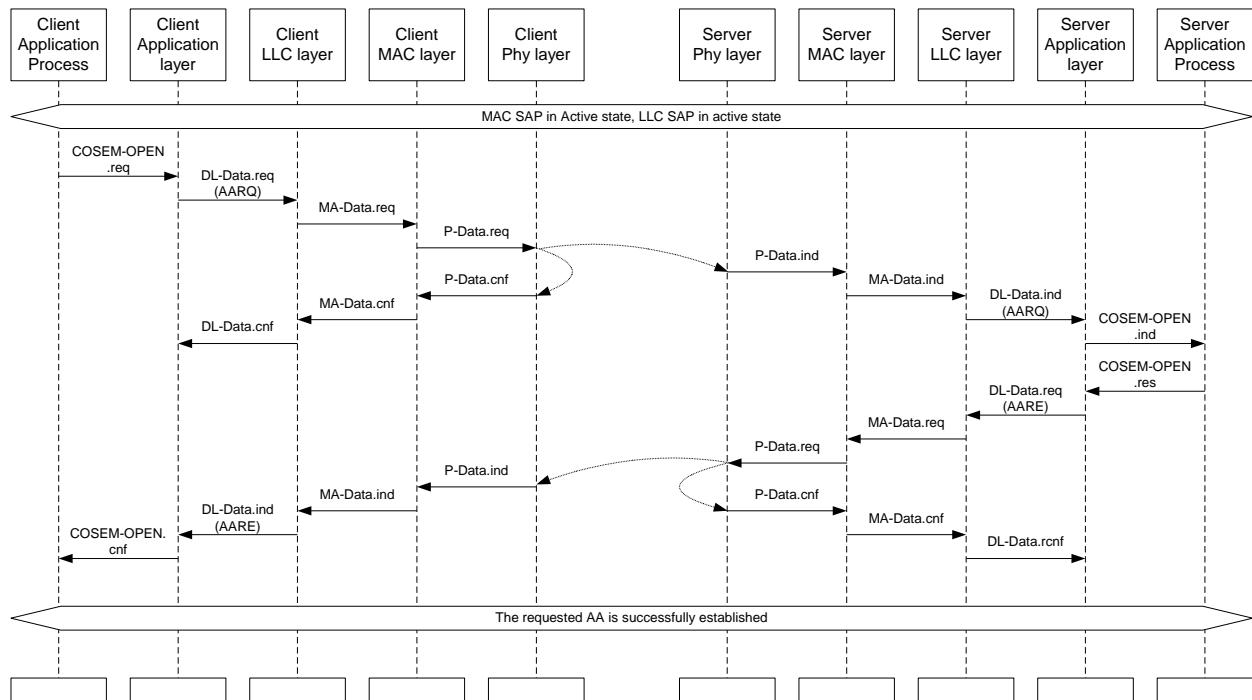


Figure 171 – MSC for the discovery and registration process

The MSC for the establishment of a confirmed AA establishment is shown in Figure 172.



**Figure 172 – MSC for successful confirmed AA establishment**

#### 10.5.7.2 Application association types, confirmed and unconfirmed xDLMS services

Table 112 shows the rules for establishing confirmed and unconfirmed AAs. In this table, grey areas represent cases, which are out of the normal operating conditions: either not allowed or have no useful purpose.

**Table 112 – Application associations and data exchange in the S-FSK PLC profile using the connectionless LLC sublayer**

Application association establishment				Data exchange	
Protocol connection parameters	COSEM-OPEN service class	Use	Type of established AA	Service class	Use
Id: LLC addresses, MAC addresses	Confirmed	Exchange AARQ/AARE APDU-s transported by DL-Data services	Confirmed	Confirmed	DL-Data services
				Unconfirmed	DL-Data services
	Unconfirmed	Send AARQ transported by DL-Data services	Unconfirmed	Confirmed (not allowed)	-
				Unconfirmed	DL-Data services

According to these:

- it is not allowed to request an xDLMS service in a confirmed way (Service\_Class = Confirmed) within an unconfirmed AA. This is prevented by the Client AL. Servers receiving such APDUs shall simply discard them, or shall send back a confirmedServiceError APDU or – if the feature is implemented – send back the optional exception-response APDU.

In this profile, the Service\_Class parameter of the COSEM-OPEN service is linked to the response-allowed parameter of the xDLMS InitiateRequest APDU. If the COSEM-OPEN service is invoked with Service\_Class == Confirmed, the response-allowed parameter shall be set to TRUE. The server is expected to respond. If it is invoked with Service\_Class == Unconfirmed, the response-allowed parameter shall be set to FALSE. The server shall not send back a response.

The Service\_Class parameter of the GET, SET and ACTION services is linked to service-class bit of the Invoke-Id-And-Priority byte. If the service is invoked with Service\_Class = Confirmed, service-class bit shall be set to 1, otherwise it shall be set to 0.

#### 10.5.7.3 xDLMS request/response type services

No specific features / constraints apply related to the use of request/response type services.

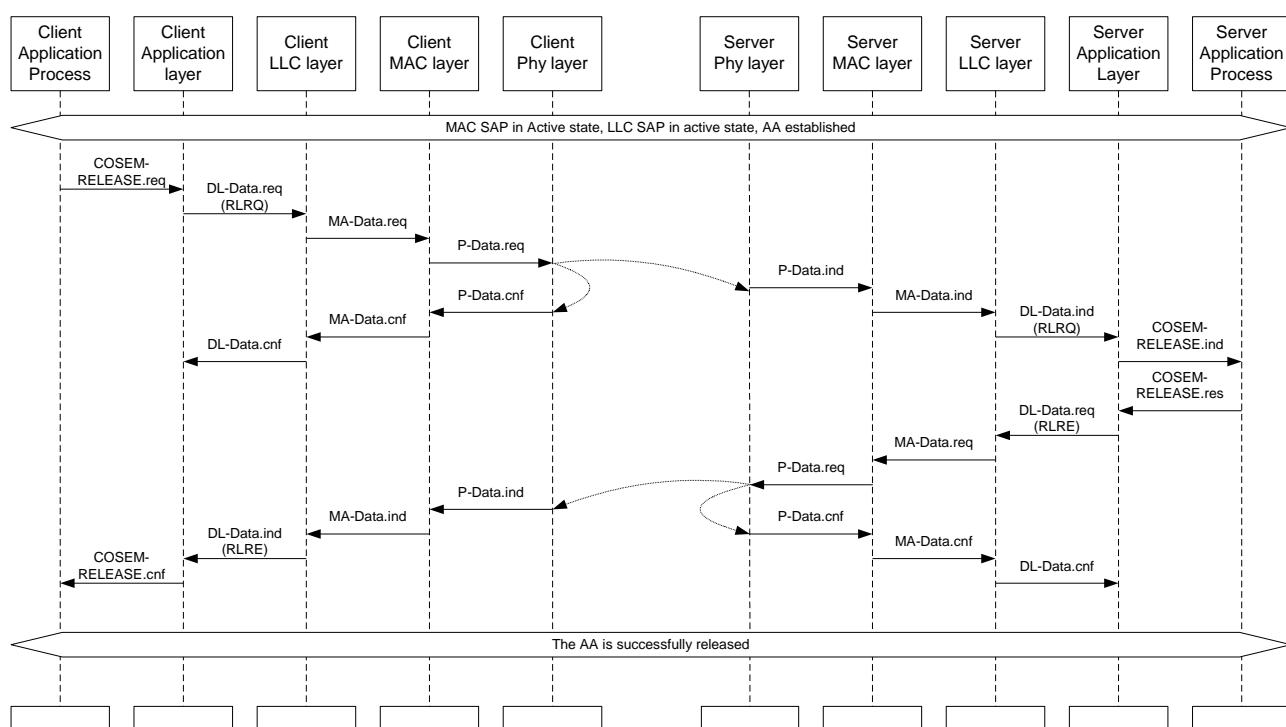
The MSC is essentially the same as for establishing confirmed AAs, except that instead of the COSEM-OPEN service primitives, the appropriate xDLMS service primitives are used.

#### 10.5.7.4 Releasing application associations

As the LLC sublayer supporting the DLMS/COSEM application layer is connectionless, the COSEM-RELEASE service may be invoked with the Use\_RLRQ\_RLRE option = TRUE to release an AA.

To secure the RLRQ APDU against denial-of-service attacks – executed by unauthorized releasing of the AA – the user-information field of the RLRQ APDU may contain the xDLMS InitiateRequest APDU, authenticated and encrypted using the AES-GCM-128 algorithm, the global unicast encryption key and the authentication key (the same as in the AARQ APDU).

The MSC for releasing an AA is shown in Figure 173.



**Figure 173 – MSC for releasing an Application Association**

#### 10.5.7.5 Service parameters of the COSEM-OPEN / -RELEASE / -ABORT services

The optional User\_Information parameters of the COSEM-OPEN / -RELEASE services are not supported in this communication profile.

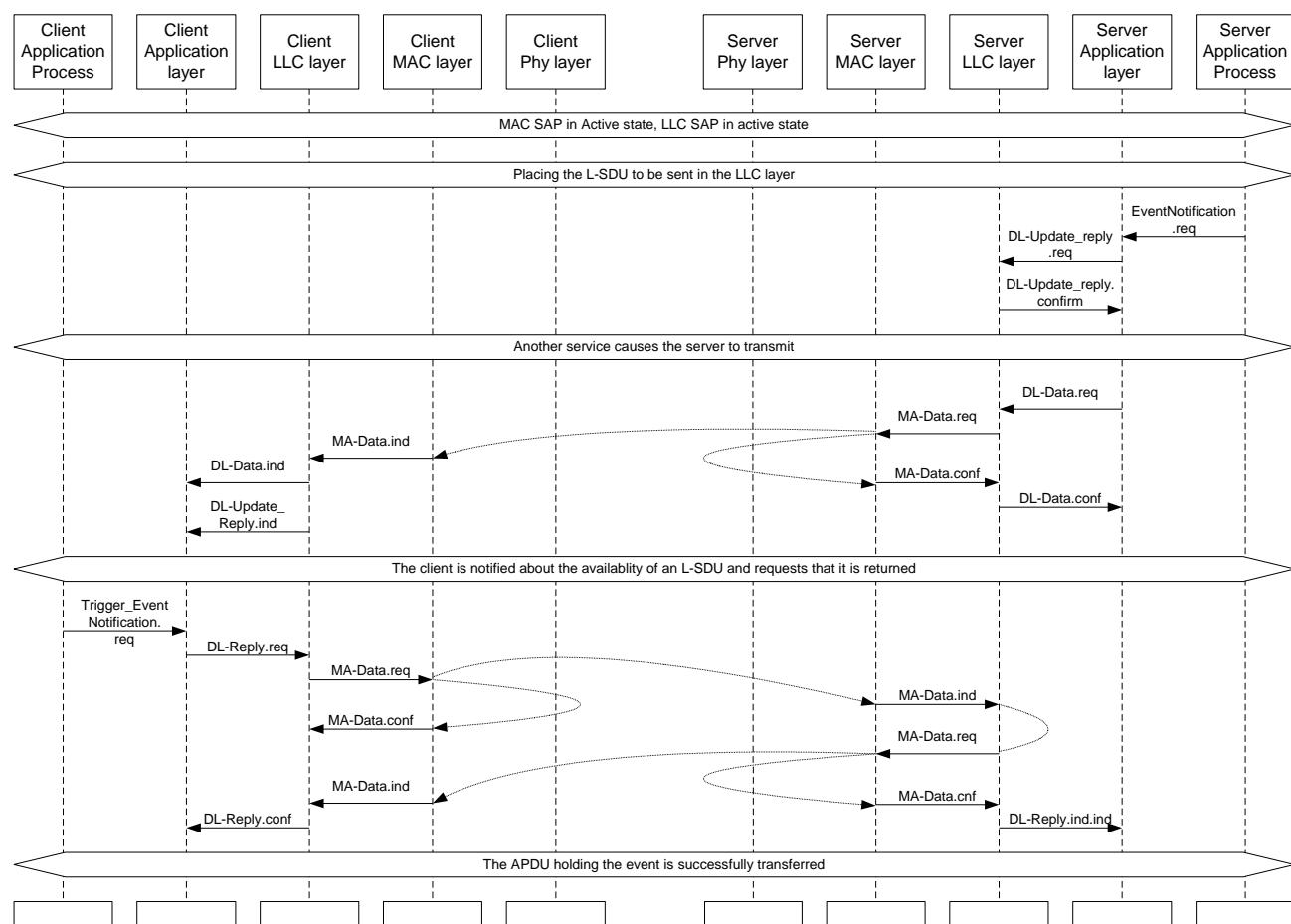
### 10.5.7.6 The EventNotification service and the TriggerEventNotificationSending service

The EventNotification (LN referencing) / InformationReport (SN referencing) services are supported by the DL-Update-Reply and DL-Reply services of the LLC sublayer:

- in the case of LN referencing, the event-notification-request APDU shall be placed into the LLC sublayer using the DL-Update-Reply.request service;
- in the case of SN referencing, the informationReportRequest APDU shall be placed into the LLC sublayer using the DL-Update-Reply.request service;
- these APDUs are available then for any client until they are cleared by placing an empty APDU.

The length of the APDUS shall not exceed the limitation imposed by the LLC / MAC sublayers.

The MSC for an EventNotification service is shown in Figure 174.



**Figure 174 – MSC for an EventNotification service**

### 10.5.7.7 Transporting long messages

In the S-FSK profile, the IEC 61334-4-32:1996 LLC sublayer imposes a limitation on the length of the APDU that can be transported. For transporting long messages, application layer block transfer is available.

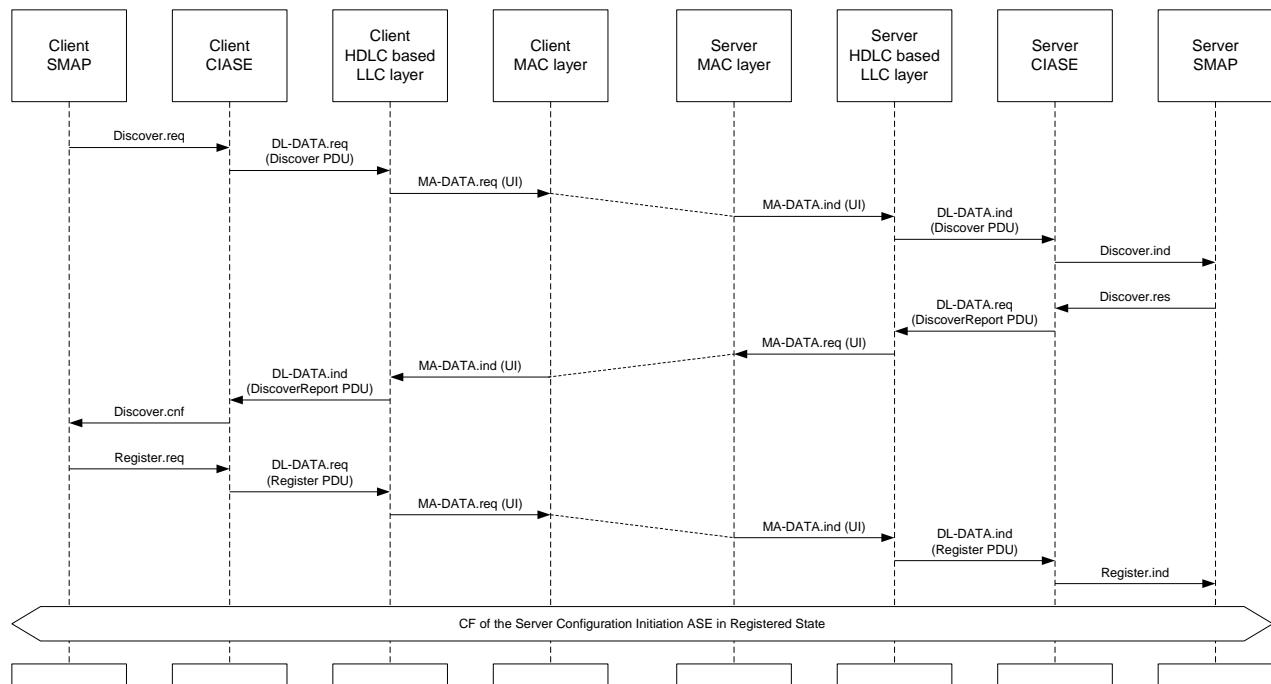
### 10.5.7.8 Broadcasting

Broadcast messages can be sent by the data concentrator, acting as a client, to servers using broadcast addresses.

### 10.5.8 Specific considerations / constraints for the HDLC LLC sublayer based profile

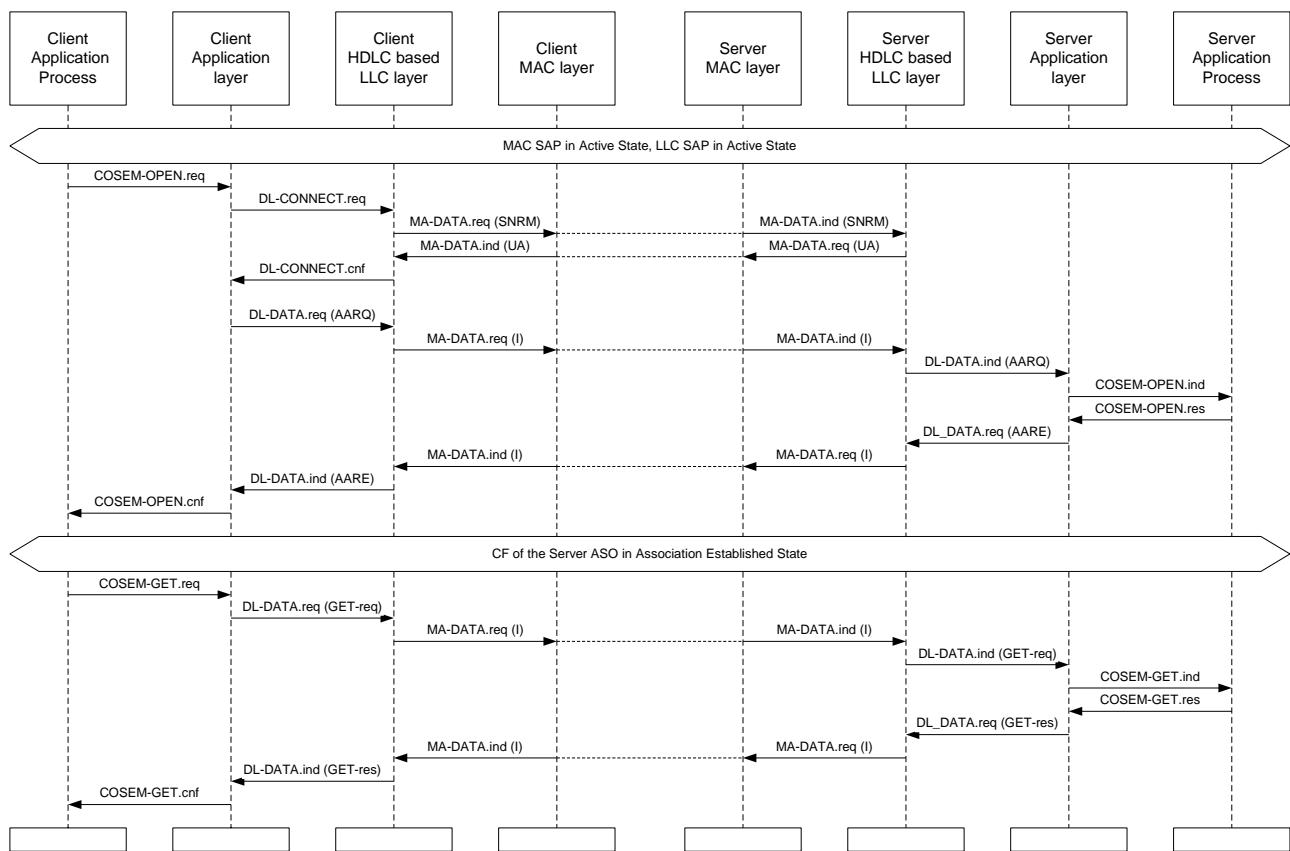
#### 10.5.8.1 Establishing application associations

AA can only be established with server systems, which have been properly registered by the initiator. The MSC for the discovery and registration process is shown in Figure 175.



**Figure 175 – MSC for the Discovery and Registration process**

The MSC for the establishment of a confirmed AA establishment is shown in the upper part of Figure 176.



**Figure 176 – MSC for successful confirmed AA establishment and the GET service**

#### 10.5.8.2 Application association types, confirmed and unconfirmed xDLMS services

10.2.6.1 applies.

#### 10.5.8.3 xDLMS request/response type services

The MSC for a COSEM GET.request service – preceded with the establishment of an AA – is shown in lower part of Figure 176.

#### 10.5.8.4 Correspondence between AAs and data link layer connections, releasing AAs

10.2.6.2 applies.

#### 10.5.8.5 Service parameters of the COSEM-OPEN/ -RELEASE/ -ABORT services

10.2.6.3 applies.

#### 10.5.8.6 The EventNotification service and protocol

10.2.6.4 applies, except that the event-notification-request APDU can be sent only when the server is registered and synchronized with the initiator.

#### 10.5.8.7 Transporting long messages

10.2.6.5 applies.

#### 10.5.8.8 Broadcasting

Broadcast messages can be sent by the data concentrator, acting as a client, to servers using broadcast addresses.

### 10.5.9 CIASE PDUs

```

CIASE pdu DEFINITIONS ::= BEGIN

CI-PDU ::= CHOICE
{
    pingRequestPDU                               [25] PingRequestPDU,
    pingResponsePDU                             [26] PingResponsePDU,
    -- reserved                                     [27]
    registerPDU                                 [28] RegisterPDU,
    discoverPDU                                  [29] DiscoverPDU,
    discoverReportPDU                           [30] DiscoverReportPDU,
    repeaterCallPDU                            [31] RepeaterCallPDU,
    clearAlarmPDU                                [57] ClearAlarmPDU

    -- CI-PDU tags are above 24, corresponding to the last unciphered DLMS APDU specified
    -- in IEC 61334-4-41.
}

-- CIASE APDUs

PingRequestPDU ::= SEQUENCE
{
    system-title-server           System-Title
}

PingResponsePDU ::= SEQUENCE
{
    system-title-server           System-Title
}

RegisterPDU ::= SEQUENCE
{
    active-initiator-system-title   System-Title,
    list-of-correspondence        Correspondence-List
}

DiscoverPDU ::= SEQUENCE
{
    response-probability          INTEGER (0..100),
    allowed-time-slots            INTEGER (0..32767),

    -- see IEC 61334-5-1 for the value of MAX_INITIAL_CREDIT
    discover-report-initial-credit  INTEGER (0..7),
    ic-equal-credit                INTEGER (0..1)
}

DiscoverReportPDU ::= SEQUENCE
{
    -- the first one of this list is the system-title of the reporting system
    system-title-list             System-Title-List,

    -- alarm-descriptor of the reporting system
    alarm-descriptor              Alarm-Descriptor OPTIONAL
}

RepeaterCallPDU ::= SEQUENCE
{
    max-adr-mac                      INTEGER (0..4095),
    nb-tslot-for-new                  INTEGER (0..255),
    reception-threshold               INTEGER (0..255) DEFAULT 104
}

ClearAlarmPDU ::= CHOICE
{
    -- clears a single alarm in all servers
    alarm-descriptor                 [0] Alarm-Descriptor,

    -- clears a list of alarms in all servers
    alarm-descriptor-list            [1] Alarm-Descriptor-List,

    -- clears a common list of alarms specified in the list of servers specified
    alarm-descriptor-list-and-server-list [2] SEQUENCE
    {
        server-id-list               System-Title-List,
        alarm-descriptor-list        Alarm-Descriptor-List
    }
}

```

```

    },

-- clears one alarm specified in each server specified
alarm-descriptor-by-server-list      [3] Alarm-Descriptor-By-Server-List
}

-- Useful types used with the S-FSK PLC profile
-- System-Title SIZE may be specified by the naming authority
System-Title ::= OCTET STRING (SIZE(8))

System-Title-List ::= SEQUENCE OF System-Title

MAC-address ::= INTEGER(0..4095)

Correspondence ::= SEQUENCE
{
    new-system-title      System-Title,
    mac-address          MAC-address
}

Correspondence-List ::= SEQUENCE OF Correspondence

Alarm-Descriptor ::= INTEGER (0..255)

Alarm-Descriptor-List ::= SEQUENCE OF Alarm-Descriptor

Alarm-Descriptor-By-Server ::= SEQUENCE
{
    server-id            System-Title,
    alarm-descriptor     Alarm-Descriptor
}

Alarm-Descriptor-By-Server-List ::= SEQUENCE OF Alarm-Descriptor-By-Server

CIASELocalError ::= ENUMERATED
{
    other                  (0),
    discover-probability-out-of-range   (1),
    discover-initial-credit-out-of-range (2),
    discoverReport-list-too-long        (3),
    register-list-too-long             (4),
    ic-equal-credit-out-of-range       (5),
    ping-no-response                  (6),
    ping-system-title-nok             (7)
}

END

```

## 10.6 The wired and wireless M-Bus profile

### 10.6.1 Scope

This subclause 10.6 specifies DLMS/COSEM wired and wireless M-Bus communication profiles for local and neighbourhood networks.

Setting up and managing the M-Bus communication channels of M-Bus devices, the M-Bus network, registering slave devices and – when required – repeaters is out of the scope of this International Standard.

The scope of this communication profile standard is restricted to aspects concerning the use of communication protocols in conjunction with the COSEM data model and the DLMS/COSEM application layer. Data structures specific to a communication protocol are out of the scope of this standard. Any project-specific definitions of data structures and data contents may be provided in project-specific companion specifications.

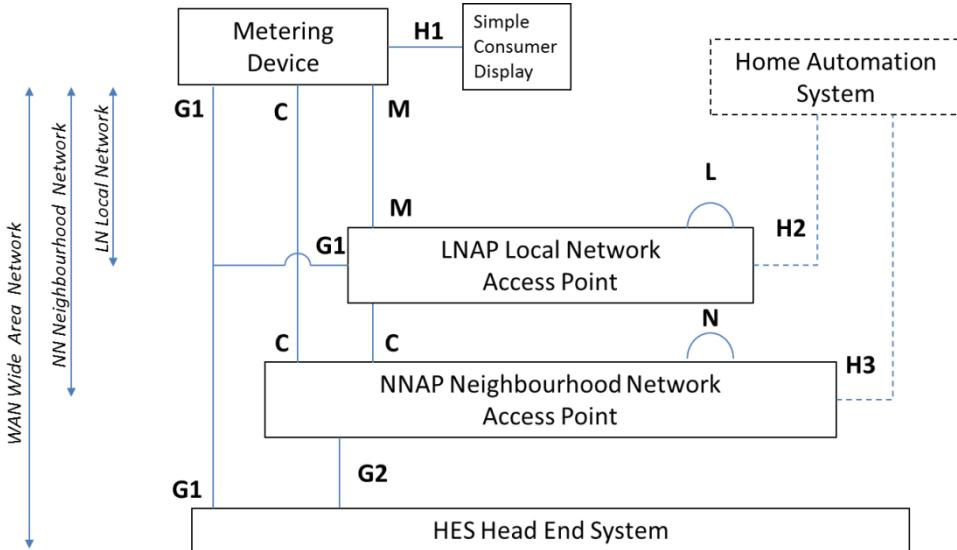
Subclause 10.6.7 provides information on M-Bus frame structures, addressing schemes and an encoding example.

Subclause 10.6.8 provides MSCs for representative instances of communication.

### 10.6.2 Targeted communication environments

In the context of the smart metering architecture introduced in IEC 62056-1-0 and shown in Figure 177, the wired and wireless M-Bus communication profiles for local and neighbourhood networks cover the following interfaces:

- the C interface between an NNAP and metering devices;
- the M interface between an LNAP and metering devices;
- the H1 interface between a metering device and a simple consumer display;
- the H2 interface between an LNAP and a home automation system.



**Figure 177 – Entities and interfaces of a smart metering system using the terminology of IEC 62056-1-0**

In all cases, metering devices act as DLMS servers.

On the C and M interface, metering devices act as M-Bus slaves. The M-Bus master is the NNAP or the LNAP.

On the H1 and H2 interfaces the metering device acts as a DLMS server. It may operate in pull mode or push mode, as M-Bus master or M-Bus slave, depending on the selection of wired or wireless M-Bus and the operating mode for wireless M-Bus.

### 10.6.3 Use of the communication layers for this profile

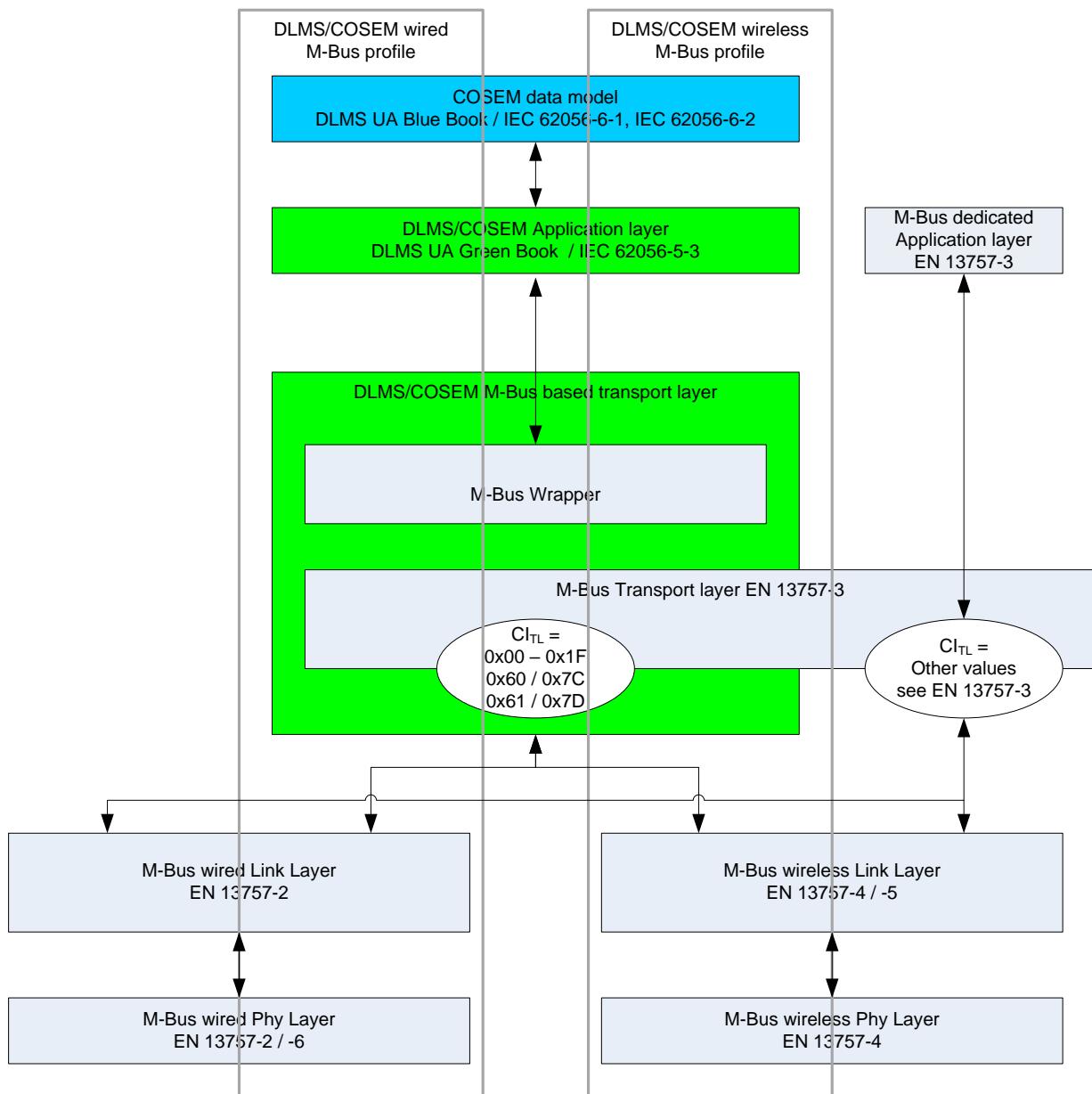
#### 10.6.3.1 Information related to the use of the standard specifying the lower layers

The DLMS/COSEM wired and wireless M-Bus communication profiles for local and neighbourhood networks use the lower-layer protocols specified in the EN 13757 series.

Clause 10.6.3.3 provides additional information on the use of the M-Bus transport layer in this communication profile.

#### 10.6.3.2 Structure of the communication profiles

The structure of the DLMS/COSEM M-Bus wired and wireless M-Bus communication profiles is shown in Figure 178.



**Figure 178 – The DLMS/COSEM wired and wireless M-Bus communication profiles**

#### 10.6.3.3 Lower protocol layers and their use

##### 10.6.3.3.1 Physical layer

The physical layer is as specified in EN 13757-2:2004 (wired, twisted pair based) and in EN 13757-4:2013 (wireless).

For battery-operated masters and/or a small number of connected meters, a wired M-Bus physical layer is specified in EN 13757-6:2015 (twisted pair based for short distances).

##### 10.6.3.3.2 Link layer

The M-Bus link layer is as specified in EN 13757-2:2004 (wired) and in EN 13757-4:2013 (wireless).

**NOTE** For wireless meter readout EN 13757-5:2015 supports simple retransmission (single-hop repeating) as well as routed wireless networks that allow extending the range of transmission.

##### 10.6.3.3.3 Transport layer

The M-Bus transport layer is as specified in EN 13757-3:2018.

Together with an M-Bus wrapper specified in 10.6.4.6, it constitutes the DLMS/COSEM M-Bus-based transport layer (TL) that acts as an adaptation layer between the link layer and the DLMS/COSEM AL.

The M-Bus TL allows several application layers to co-exist over the M-Bus lower layers. These can be:

- the M-Bus dedicated AL;
- the DLMS/COSEM AL; or
- some other AL that may be specified in the future.

The AL used is selected by the Control Information (CI) field of the M-Bus frame.

In the communication profiles specified in this standard, only the DLMS/COSEM AL is used.

There are also CI field values for network management purposes. M-Bus frames carrying such CI field values do not necessarily carry application data.

#### **10.6.3.4 Service mapping and adaptation layers**

##### **10.6.3.4.1 Overview**

As already mentioned in 10.6.3.3.3, in the wired and wireless M-Bus communication profiles for local and neighbourhood networks the DLMS/COSEM M-Bus-based TL acts as an adaptation layer between the M-Bus link layer and the DLMS/COSEM AL.

It comprises the transport layer specified in EN 13757-3:2018 and a wrapper layer.

It provides OSI-style connectionless data services with optional segmentation and reassembly to the service user DLMS/COSEM AL.

The M-Bus wrapper – specified in 10.6.4.6 – provides the addressing capability required to address client and server DLMS/COSEM APs.

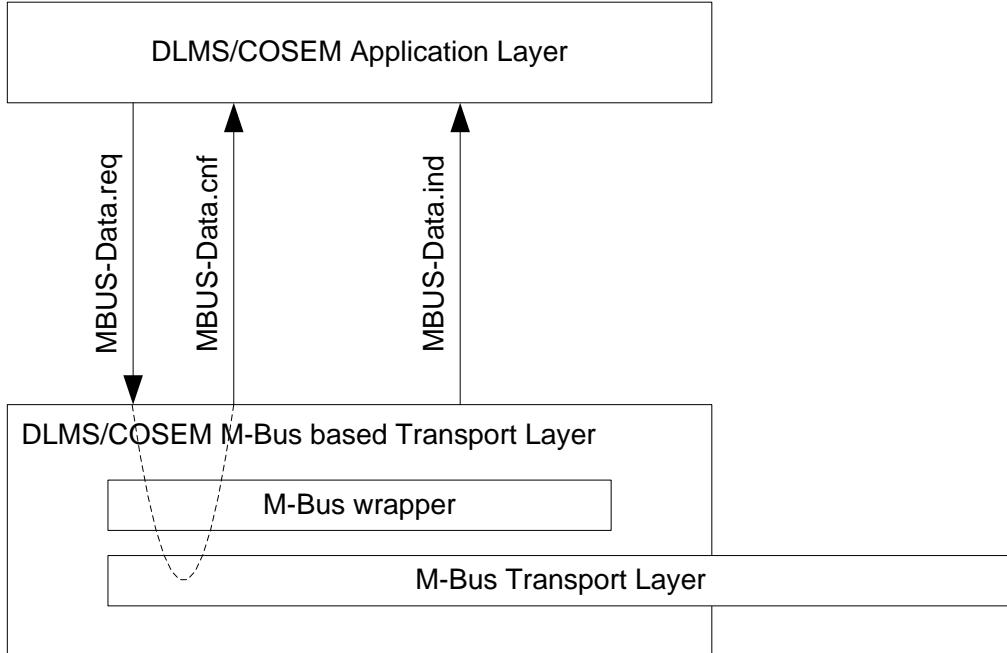
The service primitives are shown in Figure 179 and they are the same on the client and server side.

The .request service primitive is used to send a COSEM APDU to the peer TL.

The .indication service primitive indicates the reception of a COSEM APDU from the peer TL.

The .confirm service primitive is locally generated. It provides information to the AL about the status of sending the COSEM APDU.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	459/614
-----------------------	------------	-----------------------	---------



**Figure 179 – Summary of DLMS/COSEM M-Bus-based TL services**

#### 10.6.3.4.2 MBUS-DATA service primitives

##### 10.6.3.4.2.1 MBUS-DATA.request

###### 10.6.3.4.2.1.1 Function

The MBUS-DATA.request primitive is used by the DLMS/COSEM AL to request the DLMS/COSEM M-Bus-based TL to send a COSEM APDU to (a) peer DLMS/COSEM M-Bus-based transport layer(s).

NOTE Multicast or broadcasting is available only in the direction client to server.

###### 10.6.3.4.2.1.2 Semantics

The primitive shall provide the service parameters as follows:

```

MBUS-DATA.request      (
    M-Bus_Data_Header_Type,
    STSAP,
    DTSAP,
    Data
)

```

The M-Bus\_Data\_Header\_Type parameter indicates the M-Bus data header type to be used in the M-Bus frame to be sent. Its value can be *None\_M-Bus\_Data\_Header*, *Short\_M-Bus\_Data\_Header* or *Long\_M-Bus\_Data\_Header*, see Figure 182.

The STSAP parameter indicates the TL service access point (SAP) belonging to the device / AP requesting to send the Data.

The DTSAP parameter indicates the TL SAP belonging to the device(s) / AP(s) to which the Data is to be transmitted.

The Data parameter contains the COSEM APDU to be transferred to the peer AL.

###### 10.6.3.4.2.1.3 Use

The MBUS-DATA.request service primitive is invoked by either the client or the server DLMS/COSEM AL to request sending a COSEM APDU to a single peer AL or – in the case of multicast or broadcasting (by the client only) – to multiple peer ALs.

The reception of this primitive shall cause the DLMS/COSEM M-Bus-based TL to build the appropriate M-Bus data header accordingly and to construct the TPDU data unit to be passed to the M-Bus data link layer.

When the APDU to be sent is too long to fit in a single M-BUS frame, then segmentation may be used, see 10.6.3.4.3.

#### **10.6.3.4.2.2 MBUS-DATA.indication**

##### **10.6.3.4.2.2.1 Function**

The MBUS-DATA.indication primitive is used by the DLMS/COSEM M-Bus-based TL to pass a COSEM APDU received from the peer DLMS/COSEM M-Bus-based TL to the service user DLMS/COSEM AL.

##### **10.6.3.4.2.2.2 Semantics**

The primitive shall provide the service parameters as follows:

```
MBUS-DATA.indication      (
    M-Bus_Data_Header_Type,
    STSAP,
    DTSAP,
    Data
)
```

The M-Bus\_Data\_Header\_Type parameter indicates M-Bus data header type used in the M-Bus frame received. Its value can be *None\_M-Bus\_Data\_Header*, *Short\_M-Bus\_Data\_Header* or *Long\_M-Bus\_Data\_Header*, see Figure 182.

The STSAP parameter indicates the TL SAP belonging to the device / AP that has sent the Data.

The DTSAP parameter indicates the TL SAP belonging to the device / AP that has received the Data.

The Data parameter contains the COSEM APDU received from the peer AL.

##### **10.6.3.4.2.2.3 Use**

The MBUS-DATA.indication service primitive is generated by the DLMS/COSEM M-Bus-based TL to indicate to the service user DLMS/COSEM AL that a COSEM APDU from the peer layer entity has been received.

According to the received M-Bus data header, the TL allocates and passes only the M-Bus\_Data\_Header\_Type to the DLMS/COSEM AL, but not the values of the received M-Bus data header.

If the STSAP or DTSAP are not valid – meaning that there is no AA bound to the given SAPs – the message received shall be simply discarded.

NOTE If the CI<sub>TL</sub> field and the elements of the short or long M-Bus data header do not match, the TPDU is discarded by the EN 13757 M-Bus TL.

#### **10.6.3.4.2.3 MBUS-DATA.confirm**

##### **10.6.3.4.2.3.1 Function**

The MBUS-DATA.confirm service primitive allows the DLMS/COSEM M-Bus-based TL to inform the DLMS/COSEM AL about the status of transmitting the data following a .request.

#### 10.6.3.4.2.3.2 Semantics

The primitive shall provide the service parameters as follows:

```
MBUS-DATA.confirm      (
    M-Bus_Data_Header_Type
    STSAP,
    DTSAP,
    Transmission_Status
)
```

The value of the M-Bus\_Data\_Header\_Type, STSAP and DTSAP parameters shall be the same as in the .request service primitive being confirmed.

The Transmission\_Status parameter indicates the status of sending the data requested by the previous MBUS-DATA.request service. Its value can be SUCCESS, PENDING or FAILED.

#### 10.6.3.4.2.3.3 Use

The MBUS-Data.confirm service primitive is generated locally by the DLMS/COSEM M-Bus-based TL to inform the service user DLMS/COSEM AL on the status of sending the Data in the .request primitive:

- Transmission\_Status == SUCCESS means that the Data has been sent. It does not mean that the Data has been (or will be) successfully delivered to the destination;
- Transmission\_Status == PENDING means that sending the Data has been prepared or is in progress;
- Transmission\_Status == FAILED, means that the Data could not be sent by M-Bus lower layers.

### 10.6.3.4.3 MBUS-DATA protocol specification

#### 10.6.3.4.3.1 Sending COSEM APDUs

When the DLMS/COSEM AL has to send a COSEM APDU to (a) peer AL(s), it invokes the MBUS-DATA.request primitive.

Upon the reception of this request, the DLMS/COSEM M-Bus-based TL builds the appropriate TPDU. The fields of the TPDU are the following, see also Figure 182:

- the CI<sub>TL</sub> field, that indicates the kind of M-Bus data header used;
- the M-Bus data header, according to the value of the CI<sub>TL</sub> field;
- the STSAP;
- the DTSAP; and the
- Data field i.e. the COSEM APDU or – when segmentation is used – a part of it.

The value of the CI<sub>TL</sub> field depends on the M-Bus\_Data\_Header\_Type parameter:

- when M-Bus\_Data\_Header\_Type == None\_M-Bus\_Data\_Header, the value of the CI<sub>TL</sub> field shall be 0x10 when segmentation is not used, and shall be 0x00...0x1F when segmentation is used (with the FIN bit set to 0 in all segments except in the last segment);
- when M-Bus\_Data\_Header\_Type == Short\_M-Bus\_Data\_Header, the value of the CI<sub>TL</sub> field shall be 0x61 in a M-Bus frame sent by a master and 0x7D in a M-Bus frame sent by a slave;
- when M-Bus\_Data\_Header\_Type == Long\_M-Bus\_Data\_Header, the value of the CI<sub>TL</sub> field shall be 0x60 in an M-Bus frame sent by a master and 0x7C in an M-Bus frame sent by a slave.

In the case of a pull operation, the kind of M-Bus data header in M-Bus frames sent by the slave shall be the same as in the frames received from the master unless when segmentation needs to be used.

The client shall choose the M-Bus data header type according to the M-Bus media used – wired or wireless – and the capabilities of the server.

In the case of a push operation – using the xDLMS DataNotification service, see 9.3.10 – the M-Bus data header type is determined by the M-Bus slave.

The APDU can be sent without segmentation or, when it does not fit to a single M-Bus frame, with segmentation. Segmentation is available only when no M-Bus data header is used.

#### **10.6.3.4.3.1.1 Sending COSEM APDUs without segmentation**

When the M-Bus-based TL has successfully built the TPDU, it invokes the .request service primitive.

The .conf service primitive is invoked by the M-Bus-based TL with the value == SUCCESS once the lower layers confirm that the TPDU has been successfully sent.

If the lower layers cannot immediately send the M-Bus frame for whatever reason, then the .conf service primitive may be invoked by the M-Bus-based TL with the value == PENDING. When the lower layers indicate that the M-Bus frame could not be sent for whatever reason, then the .conf service primitive shall be invoked by the M-Bus-based TL with the value == FAILED.

When no M-Bus data header is used and the value of  $CI_{TL} = 0x10$  (indicating that the payload is a complete COSEM APDU) or when the short M-Bus data header ( $CI_{TL} = 0x61$ ) or the long M-Bus data header ( $CI_{TL} = 0x60$ ) is used, the TPDU has to fit in a single M-Bus frame. If the TPDU does not fit in a single M-Bus frame, then the MBUS-Data.conf service primitive shall be invoked by the M-Bus-based TL with the value == FAILED.

#### **10.6.3.4.3.1.2 Sending COSEM APDUs with segmentation**

When the M-Bus\_Data\_Header\_Type indicates None\_M-Bus\_Data\_Header and the APDU to be sent does not fit in a single M-Bus frame, the transport layer shall use segmentation. The APDU is divided by the TL into as many parts as necessary so that each segment fits in a single M-Bus frame. Each TPDU shall contain a  $CI_{TL}$  field with the appropriate value – see Figure 183 – and one segment of the COSEM APDU. The TPDU containing the next segment can be sent once the confirmation of sending the previous segment from the lower layers is received.

The .conf service primitive is invoked by the M-Bus-based TL with the value == SUCCESS when the lower layers confirm that the last segment has been successfully sent.

The MBUS-Data.conf service primitive can be invoked by the M-Bus-based TL with the value == PENDING multiple times while the segments are being sent.

#### **10.6.3.4.3.2 Receiving COSEM APDUs**

The DLMS/COSEM M-Bus-based TL uses the MBUS-DATA.indication primitive to pass the APDU received from a peer TL to the AL.

When the TL receives an M-Bus frame, it checks the value of the  $CI_{TL}$  and the STSAP/DTsap fields. When the values are not valid, the frame shall be discarded.

A STSAP and DTsap pair is valid if there is an AA bound to these SAPs.

#### **10.6.3.4.3.2.1 Receiving COSEM APDUs without segmentation**

If the  $CI_{TL}$  field indicates that the M-Bus frame contains a complete APDU, then the TL invokes the .ind service primitive.

#### **10.6.3.4.3.2.2 Receiving COSEM APDUs with segmentation**

If the  $CI_{TL}$  field indicates that the M-Bus frame contains a part of the complete APDU, then the TL assembles the Data fields received in the segments and invokes the .ind service primitive when the final segment has been received.

However, if a segment is missing, all segments shall be discarded and the .ind service primitive shall not be invoked.

NOTE The M-Bus-based TL does not provide a mechanism to recover lost segments.

### 10.6.3.5 Registration and connection management

Devices hosting DLMS servers and implementing these profiles act as M-Bus slaves.

Their installation by the M-Bus master – that may be an LNAP or an NNAP – is out of the scope of this International Standard.

**NOTE** The LNAP / NNAP may provide additional services like protocol conversion, security services and other services for entities outside the local / neighbourhood network as described in CEN / CLC / ETSI TR 50572.

The management of M-Bus entities is described in a general way in EN 13757-3:2018. Specific aspects for wired M-Bus networks are specified in EN 13757-2:2004 and for wireless M-Bus networks in EN 13757-4:2013 and EN 13757-5:2015.

The primary\_address of the device – see 10.6.4.2 and 10.6.4.3 – is held by the “DLMS server M-Bus port setup” object *primary\_address* attribute. See Clause 10.6.6.

## 10.6.4 Identification and addressing scheme

### 10.6.4.1 Overview

The wired and wireless M-Bus communication profiles for local and neighbourhood networks provide three levels of addressing:

- the Link Layer Address LLA identifies the physical device entity on the M-Bus. See 10.6.4.2, 10.6.4.3 and 10.6.4.4;
- the TL address  $CI_{TL}$  acts as a protocol selector and provides information on the structure of the TPDU and the options and capabilities available. In this profile, it selects the DLMS/COSEM M-Bus-based TL. See 10.6.4.5;
- the M-Bus wrapper, contained in the TL, provides addressing for COSEM client and server APs. See 10.6.4.6.

**NOTE 1** In some cases – e.g. in the case when wireless M-Bus is used and repeaters are present – additional addressing may be needed, but this is out of the scope of this communication profile specification.

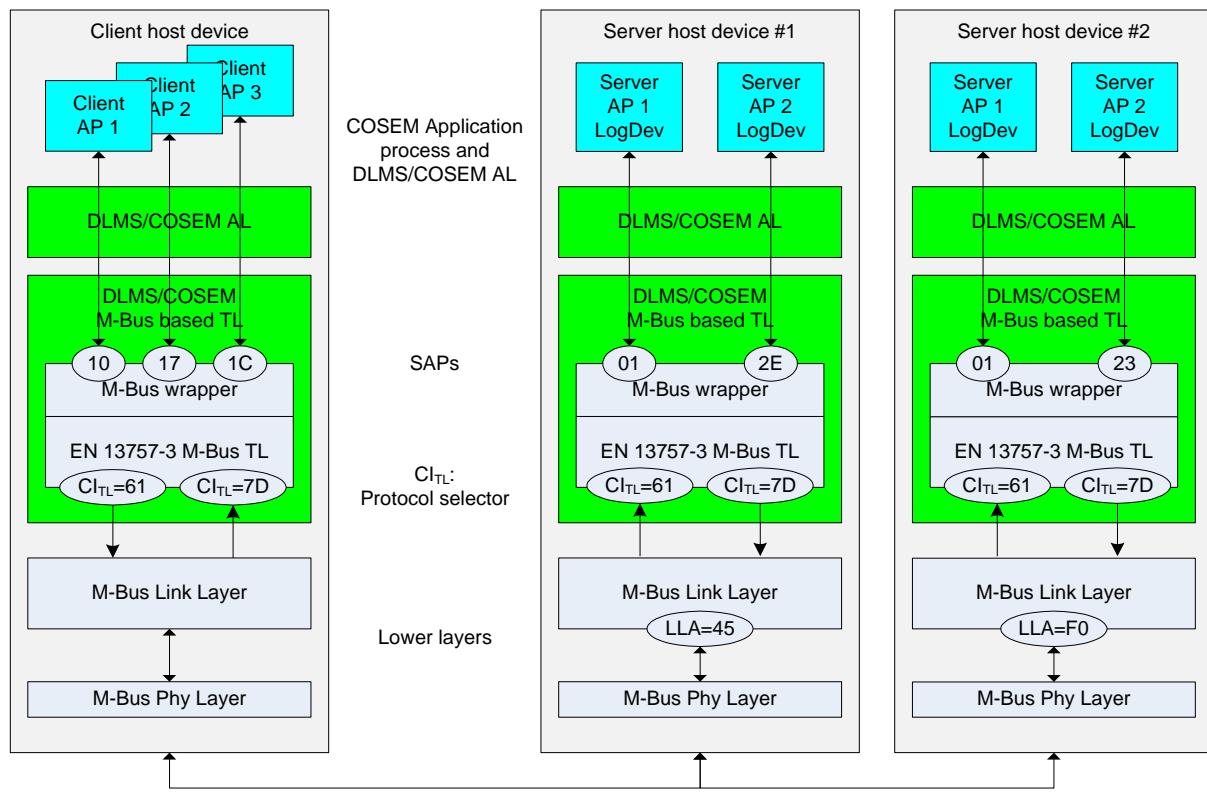
Together, these addresses allow messages to be transported non-ambiguously between a specific client AP and a specific server AP.

All addresses are provided by a protocol layer and they are present in a layer-specific M-Bus frame header. An example is shown in Figure 180.

The triplet {LLA,  $CI_{TL}$ , SAP} unambiguously identifies an AP:

- when the client AP#1 (Public Client) wants to send a xDLMS APDU to server AP#1 (Management LD) in server host device #1, the M-Bus frame shall carry the following addresses:
  - LLA = 0x45,  $CI_{TL}$  = 0x61, STSAP = 0x10, DTSAP = 0x01.
  - in the response, the M-Bus frame shall carry the following addresses:
    - LLA = 0x45,  $CI_{TL}$  = 0x7D, STSAP = 0x01, DTSAP = 0x10.

**NOTE 2** For the use of the LLA in the wired M-Bus profile, please see 10.6.4.2.



NOTE All addresses and CI codes are in hexadecimal  
LogDev = Logical device  
CI<sub>TL</sub> values indicate short M-Bus Data Header present

**Figure 180 – Identification and addressing scheme in the wired M-Bus profile**

#### 10.6.4.2 Link Layer Address for wired M-Bus

The Link Layer Address (LLA) carries, in both communication directions, the secondary station (M-Bus slave) address of the physical device entity on the M-Bus which is connected with the single addressless M-Bus master. The addressing range is as specified in Table 113.

**Table 113 – Wired M-Bus Link Layer Addresses**

Value	Description
0x00	Un-configured slaves
0x01-0xFA	Primary address <sup>a</sup> range for slaves
0xFB	Reserved for management communication with the primary master repeater
0xFC	Reserved
0xFD	Reserved for secondary addressing <sup>b</sup>
0xFE	Test and diagnostic address
0xFF	Broadcast address

NOTE Most recent EN13757 standards apply the following terms:  
<sup>a</sup> Link Layer Address (LLA) instead of primary address  
<sup>b</sup> (M-Bus) Application Layer Address (ALA) instead of secondary address

Devices connected to a wired M-Bus can support both addressing via LLA and ALA. Details of the addressing scheme for the Link Layer Address of the wired M-Bus are provided in EN 13757-2:2004, 5.7.5.

#### 10.6.4.3 Link Layer Address for wireless M-Bus

EN 13757-4:2013 specifies two data link layer frame formats:

- frame format A specified in EN 13757-4:2013, 11.3; and
- frame format B specified in EN 13757-4:2013, 11.4.

In both cases, the frames are divided into blocks, which are separated by CRC codes.

The LLA carries always the address of the sender (transmitter) wherever the frame comes from (the master or the slave). It is located in the first block.

The LLA consists of the sequence of the elements shown in Figure 181.

Manufacturer identification (M-ID)	Device Identification No (Serial No.)	Device Version (VER)	Device Type (DT)
------------------------------------	---------------------------------------	----------------------	------------------

**Figure 181 – Link Layer Address for wireless M-Bus**

If the CI<sub>ELL</sub> is present and indicates that the long Extended Link Layer Address (ELLA) is present, then the fields that follow contain the address of the receiver. It is located in the second block.

NOTE CI field CI<sub>ELL</sub> indicates usage of the Extended Link Layer with additional control service capabilities.

#### 10.6.4.4 Link Layer Address for M-Bus broadcast

In wired M-Bus networks the device capability according to the addressing via LLA and ALA determines the broadcast or multicast addressing:

- in the case of selective LLA usage, broadcast is identified by LLA = 0xFF;
- in the case of ALA usage (LLA value 0xFD), broadcast and multicast addressing is applicable.

In wireless M-Bus networks, broadcast and multicast addressing is applicable.

The addresses in ALA and in the wireless LLA are composed of the following 4 elements:

- manufacturer identification;
- device identification number;
- device version; and
- device type.

For broadcast or multicast addressing, the following rules apply (see EN 13757-3:2018):

- in the device identification number element, each individual digit can be wild-carded by a wildcard nibble 0xF;
- the manufacturer, version and device type elements can be wild-carded by a wildcard byte 0xFF.

#### 10.6.4.5 Transport layer address

An M-Bus frame may have several fields depending on the protocol layers present. Each field carries a protocol layer header and the payload is introduced by a Control Information (CI) field indicating the kind of the layer and the capabilities provided.

**Table 114 – DLMS/COSEM M-Bus-based TL CI<sub>TL</sub> values**

CI <sub>TL</sub>	Description
0x00-0x1F	No M-Bus data header is present <sup>1</sup>
0x60	Long M-Bus data header present, direction master to slave
0x61	Short M-Bus data header present, direction master to slave
0x7C	Long M-Bus data header present, direction slave to master
0x7D	Short M-Bus data header present, direction slave to master

<sup>1</sup> In this case, segmentation and reassembly is possible with restrictions.

The  $CI_{TL}$  field values introducing the DLMS/COSEM M-Bus-based TL are specified in Table 114.

It is mandatory for the server and the client to support all three CI field address structure options.

The structure of TPDU corresponding to the various  $CI_{TL}$  values is shown in Figure 182.

TPDU with no M-Bus Data Header, Data without segmentation

$CI_{TL} = 0x10$	STSAP	DTSAP	Data (xDLMS APDU)
------------------	-------	-------	----------------------

TPDU with no M-Bus Data Header, Data with segmentation, first segment

$CI_{TL} = 0x00$	STSAP	DTSAP	Data (xDLMS APDU)
------------------	-------	-------	----------------------

TPDU with no M-Bus Data Header, Data with segmentation, one segment

$CI_{TL} = 0x01..0xF$	STSAP	DTSAP	Data (xDLMS APDU)
-----------------------	-------	-------	----------------------

TPDU with no M-Bus Data Header, Data with segmentation, last segment

$CI_{TL} = 0x10..0x1F$	STSAP	DTSAP	Data (xDLMS APDU)
------------------------	-------	-------	----------------------

TPDU with short M-Bus Data Header, Data without segmentation

$CI_{TL} = 0x61 / 0x7D$	ACC   STS   CFG	STSAP	DTSAP	Data (xDLMS APDU)
-------------------------	-----------------	-------	-------	----------------------

M-Bus Short Data Header

TPDU with long M-Bus Data Header, Data without segmentation

$CI_{TL} = 0x60 / 0x7C$	ALA = Identification No   M-ID   VER   DT	ACC   STS   CFG	STSAP	DTSAP	Data (xDLMS APDU)
-------------------------	--	-----------------	-------	-------	----------------------

M-Bus Long Data Header

M-Bus\_TPDU\_formats\_OP140414.wmf

**Figure 182 – M-Bus TPDU formats**

The values  $CI_{TL} = 0x00...0x1F$  indicate that no M-Bus data header is present. In this case, the TL can provide segmentation and reassembly (see 10.6.3.4.3).

NOTE Segmentation adds less overhead than block transfer provided by the AL.

The structure of the  $CI_{TL}$  field in this case is shown in Figure 183:

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	FIN	Sequence number			

**Figure 183 –  $CI_{TL}$  without M-Bus data header**

Bits 7, 6 and 5 set to 0 indicate that no M-Bus data header is present.

Bit 4 (FIN) indicates that the data field of the TPDU carries either one part of an xDLMS APDU or the complete APDU.

Bits 3 to 0 are used for sequence numbering. The rollover of the sequence numbers is permitted, meaning that when the sequence number reaches the value 1111 and there are segments remaining to be sent, the next segment sequence number will take the value 0000.

The segmentation protocol is specified in 10.6.3.4.3.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	467/614
-----------------------	------------	-----------------------	---------

The values  $\text{CI}_{\text{TL}} = 0x61 / 0x7D$  indicate that the short M-Bus data header is present, as well as the direction of the message. Segmentation and reassembly is not available.

The values  $\text{CI}_{\text{TL}} = 0x60 / 0x7C$  indicate that the long M-Bus data header is present, as well as the direction of the message. Segmentation and reassembly is not available.

The use of the different M-Bus TPDU structures depends on the network environment as well as on the structure and capabilities of the M-Bus slave device. Therefore, their use is determined by the M-Bus slave.

The format of the TPDU in the request and the response shall be the same, except when segmentation needs to be used.

Table 115 shows values of CI fields for M-Bus link management purposes. These values are not relevant for the DLMS/COSEM M-Bus profiles, therefore they are provided for information only.

**Table 115 – CI fields used for link management purposes**

CI field	Upper layers	Description
0x80	Pure TL <sup>a</sup>	Long M-Bus data header present, to meter
0x8A	Pure TL <sup>a</sup>	Short M-Bus data header present, from meter
0x8B	Pure TL <sup>a</sup>	Long M-Bus data header present, from meter
0x8C	ELL <sup>b</sup>	Short M-Bus ELL without ELLA
0x8E	ELL <sup>b</sup>	Long M-Bus ELL with ELLA

<sup>a</sup> This CI field values indicate that the message is not intended to an application layer; no APDU is present.  
<sup>b</sup> The Extended Link Layer (ELL) is used only with wireless M-Bus.

#### 10.6.4.6 Application addressing extension – M-Bus wrapper

The DLMS/COSEM AL needs to identify the partners involved in the AA: each AA is bound to a pair of client and server SAPs. See also Figure 180 and Figure 182.

The M-Bus wrapper that constitutes a part of the DLMS/COSEM M-Bus-based TL provides the required addressing of the DLMS client and server APs. The values of the SAPs as one byte addresses on the client and the server side are specified in Table 116.

**Table 116 – Client and server SAPs**

Client SAPs	
No-station	0x00
Client Management Process	0x01
Public Client	0x10
Open for client SAP assignment	0x02...0x0F 0x11...0xFF

Server SAPs	
No-station	0x00
Management Logical Device	0x01
Reserved for future use	0x02...0x0F
Open for server SAP assignment	0x10...0x7E
All-station (Broadcast)	0xFF

## 10.6.5 Specific considerations and constraints for using certain services within profile

### 10.6.5.1 Overview

In general, there is no differentiation between wired and wireless M-Bus media for the usage of DLMS/COSEM AL services.

When using the features provided by DLMS/COSEM, the constraints set by the M-Bus medium, the use of a battery supply and the device type and configuration shall be considered by the implementer. See also 10.6.5.6.

With respect to efficiency in messaging, the following mechanism is applied to perform AA establishment and subsequent message exchange between the client and the server on the data link layer of M-Bus:

- M-Bus messages with application data in each message shall follow the scheme according to EN 13757-4:2013 where, in combination with the SND-UD2 message, the control byte code C = 0x43 allows data to be obtained instead of ACK (refer to EN 13757-4:2013, Table 24 – Function codes of the C-field in messages sent from primary stations);
- after establishment of an AA, all subsequent data exchanges follow the direct REQUEST – RESPONSE mechanism applying SND-UD2.

**NOTE** This mechanism is not applicable in combination with fragmented messages in wireless M-Bus (refer to EN 13757-4:2013, Table 24, note c).

### 10.6.5.2 Application association establishment and release: ACSE services

Table 117 summarizes the rules for establishing confirmed and unconfirmed AAs and exchanging COSEM APDUs.

**Table 117 – Application associations and data exchange in the M-Bus-based profiles**

Application association establishment				Data exchange	
Protocol connection parameters	COSEM-OPEN service class	Use	Type of established application association	Service class	Use
See below	Confirmed	Exchange AARQ/AARE APDU-s transported in M-Bus frames	Confirmed	Confirmed	M-Bus frame
				Unconfirmed	M-Bus frame
	Unconfirmed	Send AARQ-APDU-s in M-Bus frames	Unconfirmed	Confirmed (not allowed)	-
				Unconfirmed	M-Bus frame

The service parameters of the COSEM-OPEN service – see 9.3.2 – shall be used as described below.

The Protocol\_Connection\_Parameters parameter shall be:

- the protocol (profile) identifier: wired or wireless M-Bus;
- the Link Layer Address (LLA);

**NOTE** In the case of wired M-Bus, this is the LLA of the slave for both communication directions, and in the case of wireless M-Bus, this is the LLA of the sender / transmitter.

- in the case of wireless M-Bus, and when required, the Extended Link Layer Address ELLA;
- the M-Bus data header type, and when present, the short or long M-Bus data header;
- server logical device address: COSEM logical device SAP;
- client\_Id: COSEM client SAP.

Any server (destination) address parameter may contain special addresses (All-station, No-station, etc.).

The User\_Information service parameter is not used.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	469/614
-----------------------	------------	-----------------------	---------

The Service\_Class parameter shall be used as follows:

- in the M-Bus-based profiles, the Service\_Class parameter of the COSEM-OPEN service is linked to the response-allowed parameter of the xDLMS InitiateRequest APDU:
  - if the COSEM-OPEN service is invoked with Service\_Class == Confirmed, the response-allowed parameter shall be set to TRUE. The server is expected to respond;
  - if it is invoked with Service\_Class == Unconfirmed, the response-allowed parameter shall be set to FALSE. The server shall not send back a response.
- unconfirmed AAs between a client and a group of logical devices are established using a COSEM-OPEN service with Service\_Class == Unconfirmed and a group of logical device addresses (for example broadcast address).

As the lower layers are connectionless, the Use\_RLRQ\_RLRE service parameter of the COSEM-RELEASE service – see 9.3.3 – shall be set to TRUE.

### **10.6.5.3 xDLMS services**

#### **10.6.5.3.1 Request – response type services**

The Service\_Class parameter of the GET, SET, ACTION and ACCESS services is linked to the service class bit of the Invoke-Id-And-Priority / Long-Invoke\_Id-And-Priority field. If the service is invoked with Service\_Class = Confirmed, the corresponding service class bit shall be set to 1, otherwise it shall be set to 0.

It is forbidden to request an xDLMS service in a confirmed way (Service\_Class = Confirmed) within an unconfirmed AA established on the top of the DLMS/COSEM M-Bus-based transport layer. This is also prevented by the Client AL. Servers receiving such APDUs shall simply discard them, or shall send back a ConfirmedServiceError APDU or, if the feature is implemented, send back the optional ExceptionResponse APDU.

#### **10.6.5.3.2 Unsolicited services**

Application of the DLMS/COSEM unsolicited services (EventNotification, DataNotification, and InformationReport) in M-Bus networks depends on capability and configuration of M-Bus entities and should therefore be specified in project-specific companion specifications.

#### **10.6.5.3.3 Broadcast messages**

For wired M-Bus the use of broadcast messages by addressing with primary address (= link layer address, equal to 255) is supported. Due to the fact that broadcast messages cannot be acknowledged on the link layer level, a subsequent verification on the application level may need to be performed.

In general also broadcast and multicast for secondary addresses are available using the wildcard mechanism in the address parts (EN 13757-3:2018, referenced from EN 13757-4:2013).

Therefore, in wireless M-Bus networks broadcast / multicast messages are applicable, taking into account the fact that from a DLMS viewpoint, broadcast of COSEM APDUs via wireless M-Bus is available if supporting M-Bus lower layers will offer corresponding functionality.

### **10.6.5.4 Security mechanisms**

The security mechanism specified in 9.2 and DLMS UA 1000-1 may be used without specific constraints.

### **10.6.5.5 Transporting long application messages**

There are two mechanisms available to transport long messages that do not fit directly in a single M-Bus frame:

- segmentation provided by the TL, see 10.6.3.4.3 and 10.6.4.5. This mechanism is available only when no M-Bus data header is present in the TPDU (refer to EN13757-1:2014);
- DLMS/COSEM application layer block transfer. This mechanism can be used with all TPDU structures.

470/614	2021-12-21	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

### 10.6.5.6 Media access, bandwidth and timing considerations

For DLMS/COSEM unsolicited services – see 10.6.5.3.2 – the M-Bus alarm messaging in wired and wireless M-Bus installations could get certain delays due to:

- out of transmission credits<sup>9</sup> for battery-operated devices;

NOTE Credits as information depends on project-specific companion specifications: these are not free parameters due to complex embedding and therefore accessible in application.

- waiting time for access until next transmission (wireless M-Bus only);
- disturbance on radio channel (wireless M-Bus only);
- handling message priority is not supported by M-Bus.

For broadcast in wireless M-Bus networks, some further aspects are important:

- energy consumption depends on various parameters, device and system design which, in general, does not prevent using broadcast;
- accessibility is essential for broadcast and multicast, where EN 13757-4:2013, Table 27 indicates different accessibility (no, temporary, limited, unlimited) with device examples;
- mains-powered meters and communication modules are accessible without limitations (with respect to duty cycles!). On the other hand, battery-powered meters pose some restrictions (see transmission credits), but may operate with dedicated time windows to receive messages.

### 10.6.6 Communication configuration and management

The COSEM interface classes that apply to the communication profiles specified in this International Standard are the following:

- DLMS server M-Bus port setup (class\_id = 76), see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.8.6,
- M-Bus slave port setup (class\_id = 25), see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.8.2;
- wireless mode Q channel (class\_id = 73), see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.8.4;
- M-Bus diagnostic (class\_id = 77), see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.8.7.

### 10.6.7 M-Bus frame structures, addressing schemes and examples

#### 10.6.7.1 General

This subclause 10.6.7 gives an overview and examples of M-Bus frame structures and related addressing schemes for wired and wireless M-Bus.

For transporting xDLMS APDUs – the Data – only the source and destination SAPs providing DLMS/COSEM application process addressing are relevant. All other fields of the M-Bus frame are processed by the transport layer and the lower layers. For this reason, this subclause 10.6.7 is informative.

Further details and related usage context are to be found in the appropriate standards referenced.

In M-Bus frames there may be one or more Control Information (CI) fields present that provide information about the protocol layers and the related M-Bus frame fields present as well as the usage of those fields. All CI-Fields<sup>10</sup> relevant for the DLMS/COSEM M-Bus communication profiles are specified in clause 10.6.4.5.

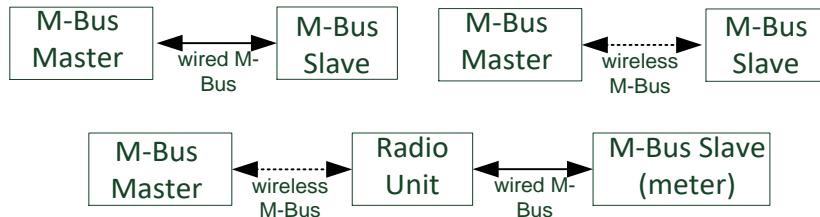
<sup>9</sup> The battery-operated meter has to protect itself from frequent periodical readout in context with available communication budget. For this constraint, such a type of meter limits the number of transactions by credits. If the credit is used up, the communication (access) will stop until the device receives the next credit.

<sup>10</sup> Further CI-Fields are specified for management purposes, see EN 13757-3:2018 and EN 13757-7:2018

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	471/614
-----------------------	------------	-----------------------	---------

In general, the M-Bus frame structure is determined by the master in the case of wired M-Bus and by the slave in the case of wireless M-Bus. The selection of the appropriate M-Bus frame structure depends also on a number of other factors:

- the communication path: direct or cascaded (e.g. with conversion unit between wired and wireless M-Bus), see Figure 184;
- the physical structure of the M-Bus slave (e.g. with integrated or with external radio unit), see e Figure 184;
- the operation phase: installation or normal operation;
- the installation process, e.g. manual or automatic address assignment.



**Figure 184 – M-Bus communication paths direct or cascaded**

In the following sections, M-Bus frame structures and addressing-related information are described and examples are shown for:

- wired M-Bus:
  - Figure 185 – Wired M-Bus frame structure, none M-Bus data header;
  - Figure 186 – Wired M-Bus frame structure with long M-Bus data header;
- wireless M-Bus:
  - Figure 187 – Wireless M-Bus frame structure with short ELL, no M-Bus data header;
  - Figure 188 – Wireless M-Bus frame structure with long ELL, no M-Bus data header;
  - Figure 189 – Wireless M-Bus frame structure with long ELL and long M-Bus data header.

#### 10.6.7.2 None, short or long M-Bus data header

##### 10.6.7.2.1 Wired M-Bus

The wired M-Bus is a master/slave bus, which connects a single master with several slaves.

The master may address the slave either by the Link Layer Address, LLA (which is applied in the link layer of master message) or by the Application Layer Address, ALA. Due to the limited range of Link Layer Addresses – see Table 113 – the LLA must be assigned during the installation of the slave. This can be done manually by a service technician or automatically from the master.

The ALA shall be assigned by the manufacturer and it shall be unique worldwide (however, an operator may change it within a system).

For addressing via the ALA, the master has to send a special Select-Command which contains the ALA of the slave. Such Select-Commands may apply wildcards. In this case no, one or several slaves will be selected. Wildcards can be used to search (test) for slaves connected to the M-Bus (refer to Wild Card Search in EN 13757-7:2018). Once a slave is detected, the master can assign a unique LLA to it.

This justifies why the slave must apply the ALA in the long transport layer in the slave response for an automatic assignment by the master.

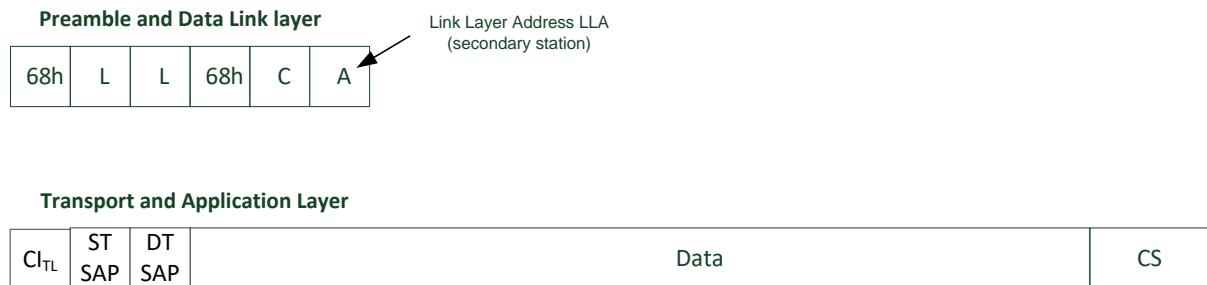
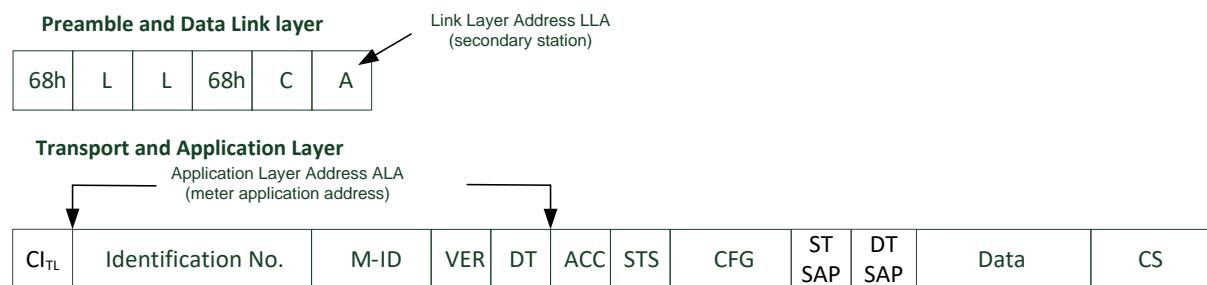
For a manually performed assignment, a frame with a simple wrapper in the slave response will be sufficient. The master should in general apply frames with simple wrappers.

NOTE 1 There will be the option to disable the long TL after the address assignment process.

Figure 185 and Figure 186 show wired M-Bus frame structures according to EN 13757-2:2004 / IEC 60870-5-1:1990.

## Legend to the Figures:

L	L-Field, length field, number of user data octets
C	C-field, control field
A	Primary address (assigned during installation process), 1 octet
Identification No	Device identification number (serial number, Serial-nr), 4 octets
M-ID	Manufacturer ID, 2 octets
VER	Version, 1 octet
DT	Device Type, 1 octet
CI <sub>TL</sub>	CI-Field for TL
ACC	Access number
STS	Status
CFG	Configuration field
CS	Check sum
STSAP	Source Transport Service Access Point
DTSAP	Destination Transport Service Access Point
Address elements (ALA)	
M-ID	Manufacturer ID, 2 octets
Identification No	Device identification number (serial number, serial-nr), 4 octets
VER	Version, 1 octet
DT	Device Type, 1 octet

**Figure 185 – Wired M-Bus frame structure, none M-Bus data header****Figure 186 – Wired M-Bus frame structure with long M-Bus data header**

NOTE 2 The Link Layer Address (LLA) contains always the secondary station address (slave). The Application Layer Address (ALA) contains always the meter application address (M-Bus address of server entity address) and is available only in frames with long M-Bus data header.

**10.6.7.3 Wireless M-Bus****10.6.7.3.1 Extended Link Layer**

For the communication via wireless M-Bus the short or long Extended Link Layer shall be used in each message transmitted.

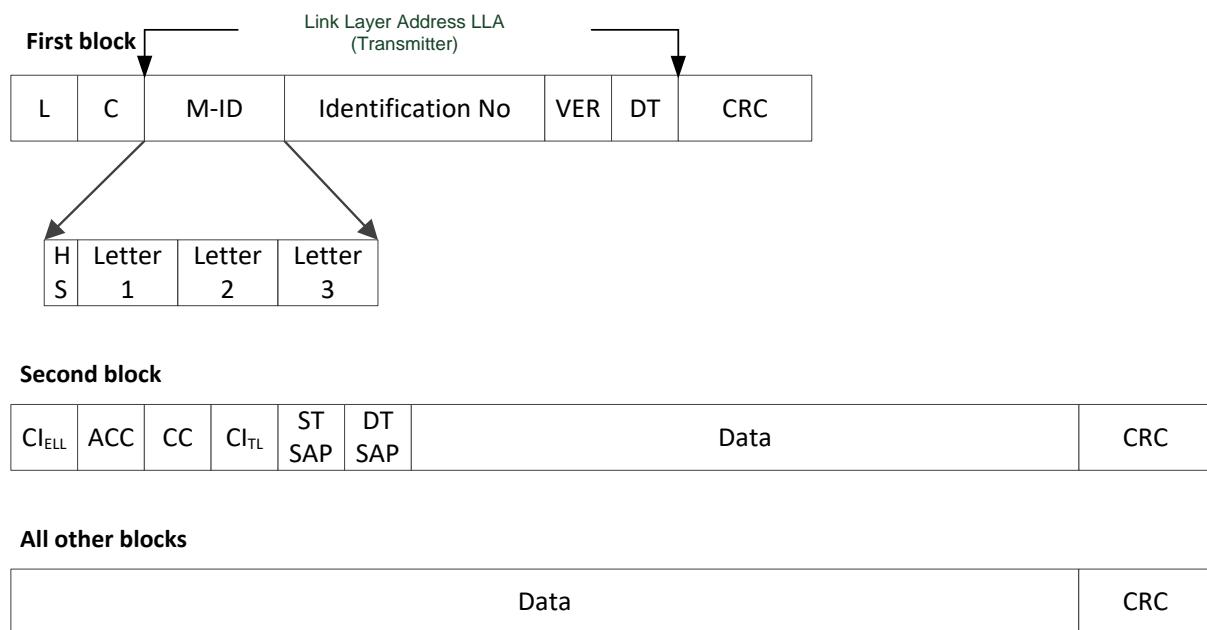
The wireless M-Bus uses the access Number (ACC) for the identification of an old or new M-Bus frame types, see EN 13757-7:2018, 7.5.5. Therefore, the wireless M-Bus protocol shall provide an access number for both the master and the slave. Additional link control bits are required which, for example, signal the availability of the server. Different services are provided by the Extended Link Layer type; see EN 13757-4:2013, 12.2.

Battery-operated slaves cannot provide unlimited availability for the master. That's why the access time is controlled by the slave itself. For that reason, the slave transmits periodically unrequested messages. The master may access the slave after such a transmission. For all frames transmitted between the master and the server, the LLA and the ELLA shall be applied. For the unrequested transmission by the slave, only the LLA is required.

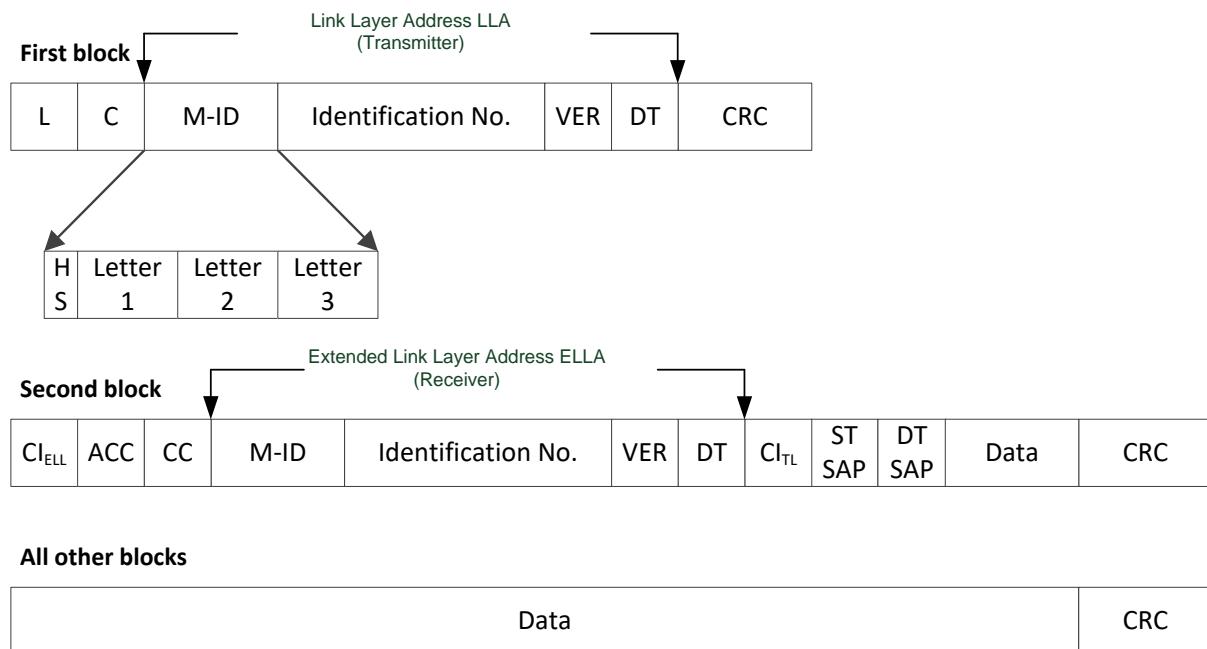
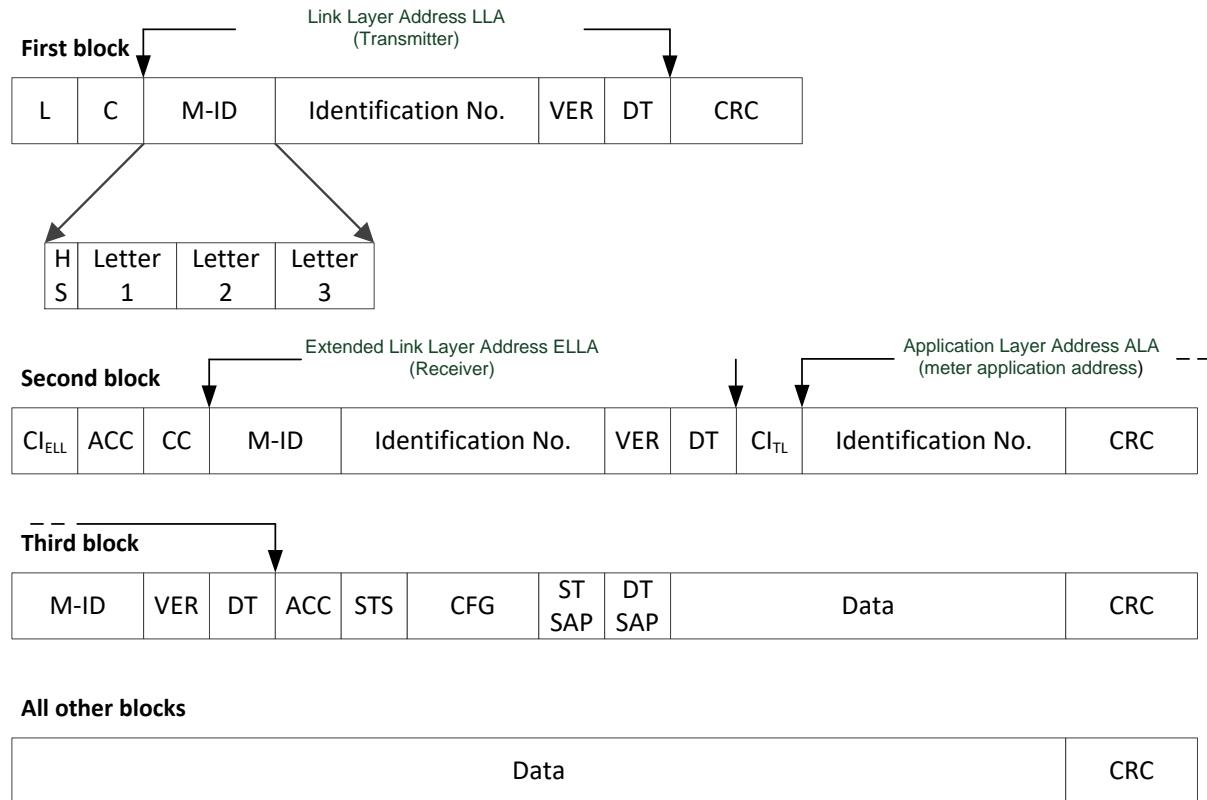
Figure 187, Figure 188 and Figure 189 show wireless M-Bus frame structures according to EN 13757-4:2013 and apply frame format A as specified in EN 13757-4:2013, 11.3.

Legend to the Figures:

L	L-Field, length field, number of user data octets
C	C-field, control field
CRC	Cyclic redundancy check
CI	CI-Field for Extended Link Layer $CI_{ELL}$ or TL $CI_{TL}$
ACC	Access number
CC	Communication Control Field
STS	Status
CFG	Configuration field
STSAP	Source Transport Service Access Point
DTSAP	Destination Transport Service Access Point
Address elements (LLA, ELLA, ALA)	
HS	Selector for hard-coded (by manufacturing process) or soft-coded (by installation process) address
M-ID	Manufacturer ID, 2 octets
Identification No	Device identification number (serial number, Serial-nr), 4 octets
VER	Version, 1 octet
DT	Device Type, 1 octet



**Figure 187 – Wireless M-Bus frame structure with short ELL, no M-Bus data header**

**Figure 188 – Wireless M-Bus frame structure with long ELL, no M-Bus data header****Figure 189 – Wireless M-Bus frame structure with long ELL and long M-Bus data header**

NOTE The Link Layer Address (LLA) always contains the transmitter address (radio unit). The Extended Link Layer Address (ELLA) always contains the receiver address.

The ELLA is available only in frames with a long Extended Link Layer.

The Application Layer Address (ALA) always contains the meter application address (M-Bus address of the slave entity). The ALA is available only in frames with CI-Fields indicating an M-Bus long data header.

DLMS User Association	2021-12-21	DLMS UA 1000-2 Ed. 11	475/614
-----------------------	------------	-----------------------	---------

Frames without or with short transport layers are applied only when the LLA is sufficient.

The link layer address (LLA + ELLA) is a hard-coded address assigned by the manufacturer and shall not be changeable by the operator. In the event that an external communication adapter is used, the ALA needs to be assigned to this adapter.

#### 10.6.7.3.2 Transport layer

When several slaves share one radio unit, the ELLA is not sufficient for an unambiguous addressing of the dedicated slave. In this case, the long transport layer should be used. The ALA contains the intended slave address.

Otherwise, the wrapper with the CI-Field value 0x10 may be used (but no segmentation).

#### 10.6.7.4 Encoding example: Data-Notification carrying daily billing data

##### 10.6.7.4.1 Overview

This subclause of the Annex shows an encoding example for sending daily billing data by the server using the “Compact data” interface class. The data are pushed using the DataNotification service – see 9.3.10 – via wireless M-Bus using frame format A.

The “Compact data” interface class – see DLMS UA 1000-1 – allows the separation of the metadata from the data. This drastically reduces the overhead which is essential for battery-operated devices and for restrained communication media, such as wireless M-Bus.

The example also shows the encoding of the information related to the lower layers.

##### 10.6.7.4.2 Example: Daily billing data

Table 118 shows the daily billing data that are captured – together with the *template\_id* – to the *compact\_buffer* attribute of a “Compact data” object.

**Table 118 – Example: Daily billing data**

Data	class_id	Logical name	attribute_id	Size (bytes)	Type
Template_id	62	0-0:66.0.0.255	4	1	unsigned
Unix time	1	0-0:1.1.0.255	2	4	double-long-unsigned
Operating status	1	0-0:96.5.0.255	2	1	unsigned
Error register	1	0-0:97.97.0.255	2	1	unsigned
Total index	3	7-0:13.83.1.255	2	4	double-long-unsigned
Index F1	3	7-0:13.83.1.255	2	4	double-long-unsigned
Index F2	3	7-0:13.83.1.255	2	4	double-long-unsigned
Index F3	3	7-0:13.83.1.255	2	4	double-long-unsigned
Activity calendar name	20	0-0:13.0.0.255	2	6	octet-string
Event counter	1	0-0:96.15.1.255	2	2	long-unsigned

Here, the overhead is only 1 byte, identifying the template.

Figure 190 shows the example of transporting the Data-Notification APDU without ciphering and with authenticated encryption.

Examples Daily billing data (by data notification) without DLMS/COSEM security				Daily billing data (by data notification) with DLMS/COSEM security option encoding and authentication			
Field ID	Description	Length (octets)	Layer	Field ID	Description	Length (octets)	Layer
L Field	Length of data	1		L Field	Length of data	1	
C Field	Send/NoReply	1		C Field	Send/NoReply	1	
M Field (Transmitter)	Manufacturer code	2		M Field (Transmitter)	Manufacturer code	2	
A Field	Ident No	4		A Field	Ident No	4	
A Field	Version	1		A Field	Version	1	
A Field	Device type	1		A Field	Device type	1	
CRC	Checksum block	2		CRC	Checksum block	2	
CI_ELL Field	Extended Link Layer (long)	1		CI_ELL Field	Extended Link Layer (long)	1	
CC	Communication Control	1		CC	Communication Control	1	
ACC	Access Counter	1		ACC	Access Counter	1	
M Field (Receiver)	Manufacturer code	2		M Field (Receiver)	Manufacturer code	2	
A Field	Ident No	4		A Field	Ident No	4	
A Field	Version	1		A Field	Version	1	
A Field	Device type (Communication controller)	1		A Field	Device type (Communication controller)	1	
CI_TL Field	No header	1		CI_TL Field	No header	1	
SAP	STSAP	1		SAP	STSAP	1	
SAP	DTsap	1		SAP	DTsap	1	
Service	DataNotification	1		Cyphering Service	Gen Glo/Ded Cyphering Service	1	
Service	Long-Invoke-Id-And-Priority Part 1	1		SystemTitle	Part 1	1	
CRC	Checksum block	2		CRC	Checksum block	2	
Service	Long-Invoke-Id-And-Priority Part 2	3		SystemTitle	Part 2	7	
Service	date-time	7		Cyph Service	Tag for ciphered content	1	
Notification body	Structure with 1 element	2		Cyph Service	Length of ciphered content	1	
Payload	Octet-string and length	2		Security Header	Security Control Byte SC-AE	1	
Payload	Template-ID	1		Security Header	Invocation Counter	4	
Payload	Unix time Part 1	1		Service	DataNotification	1	
CRC	Checksum block	2		Service	Long-Invoke-Id-And-Priority Part 1	1	
Payload	Unix time Part 2	3		CRC	Checksum block	2	
Payload	Operating status	1		Service	Long-Invoke-Id-And-Priority Part 2	3	
Payload	Error register	1		Service	date-time	7	
Payload	Total index	4		Notification body	Structure with 1 element	2	
Payload	Index F1	4		Payload	Octet-string and length	2	
Payload	Index F2 Part 1	3		Payload	Template-ID	1	
CRC	Checksum block	2		Payload	Unix time Part 1	1	
Payload	Index F2 Part 2	1		CRC	Checksum block	2	
Payload	Index F3	4		Payload	Unix time Part 2	3	
Payload	Activity calendar name	7		Payload	Operating status	1	
Payload	Event counter	2		Payload	Error register	1	
CRC	Checksum block	2		Payload	Total index	4	
Number of octets				Payload	Index F1	4	
82				Payload	Index F2 Part 1	3	
				CRC	Checksum block	2	
				Payload	Index F2 Part 2	1	
				Payload	Index F3	3	
				Payload	Activity calendar name	7	
				Payload	Event counter	2	
				Authentication TAG	MAC Part 1	3	
				CRC	Checksum block	2	
				Authentication TAG	MAC Part 2	9	
				CRC	Checksum block	2	
				Number of octets			
				113			

**Figure 190 – Daily billing data without / with DLMS/COSEM security applied**

### **10.6.8 Message sequence charts**

This subclause 10.6.8 shows some message sequence charts for communication between a DLMS client and server using the wired and wireless M-Bus profiles. They are representative for all CI value address structure options according to headers and wired/wireless connection.

Figure 191 shows the MSC for the COSEM-OPEN service for wired M-Bus without M-Bus header.

Figure 192 shows the MSC for the GET service for wired M-Bus without M-Bus header.

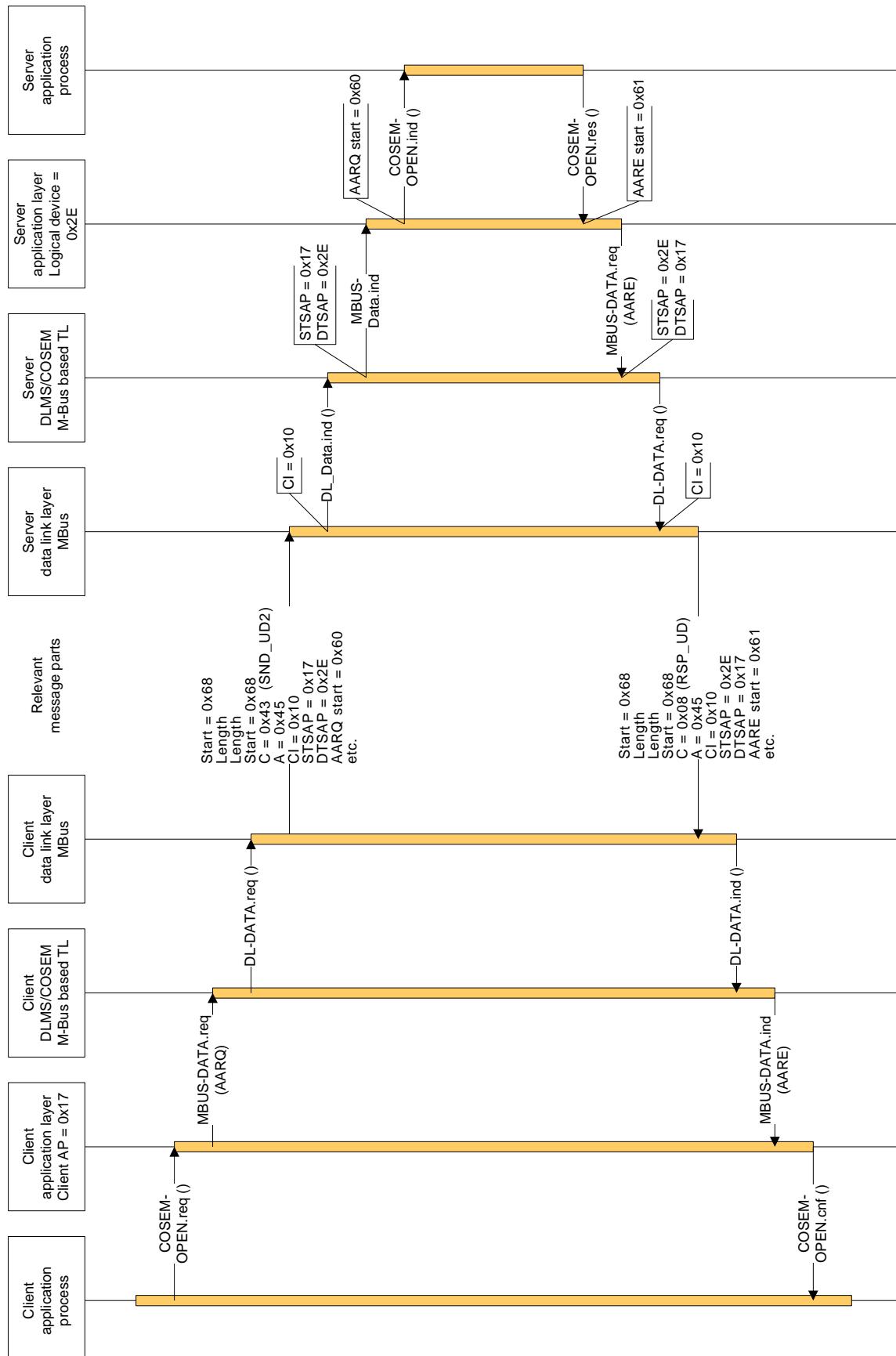


Figure 191 – MSC for the COSEM-OPEN service for wired M-Bus, none M-Bus header

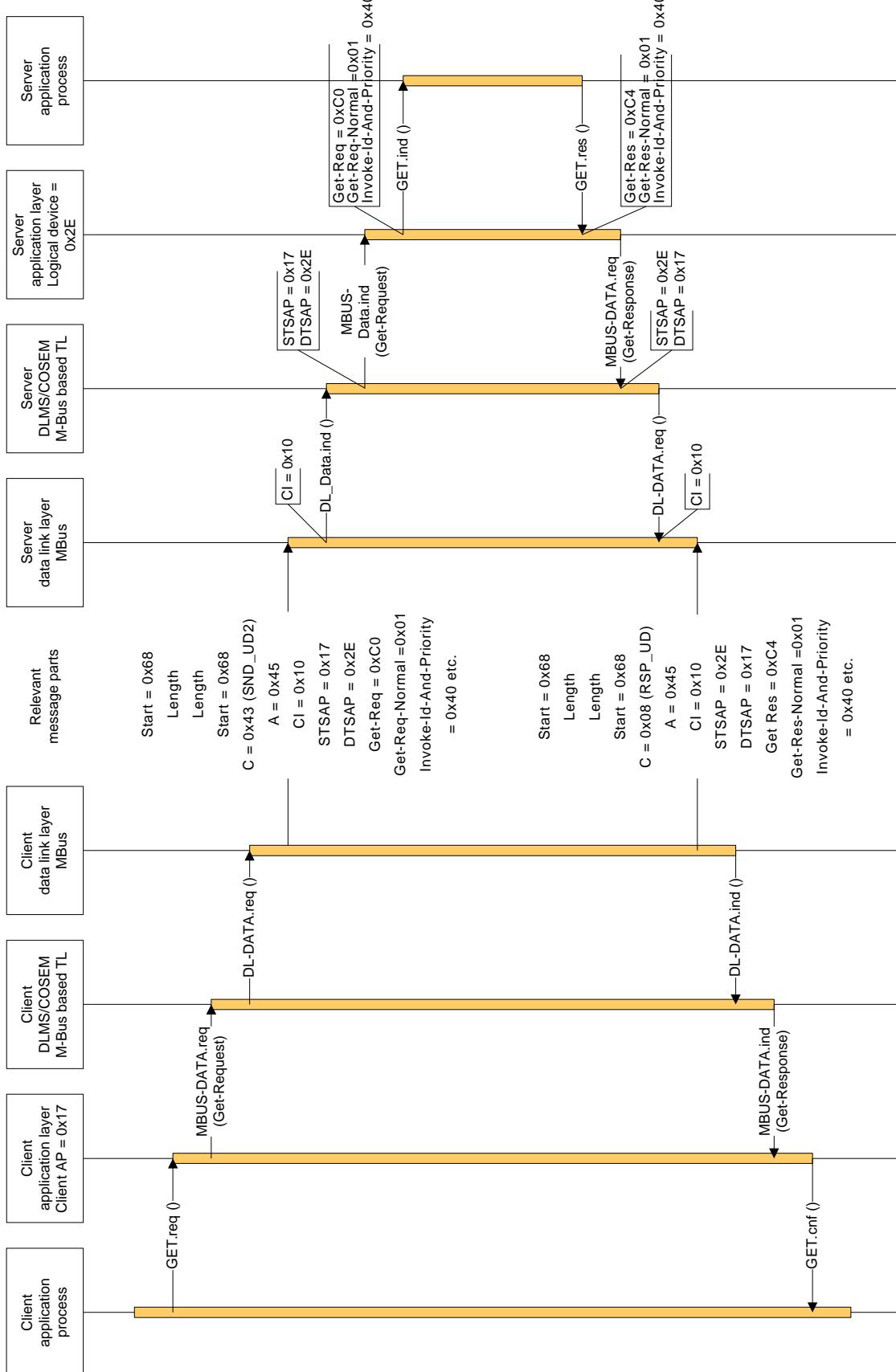
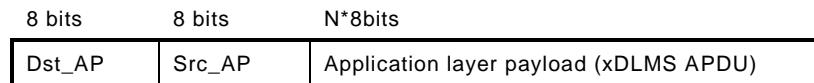


Figure 192 – MSC the GET service for wired M-Bus, none M-Bus header

## 10.7 SMS short wrapper

This subclause specifies the transport of COSEM APDUs in an SMS.

The payload of an SMS message is the COSEM APDU prepended with the identifier of the Destination\_AP and the Source\_AP as shown in Figure 193:



- Dst\_AP = Destination AP identifies the destination Application Process;
- Src\_AP = Source AP identifies the source Application Process.

**Figure 193 – Short wrapper**

Table 119 specifies the identifiers of reserved Application Processes:

**Table 119 – Reserved Application Process SAPs**

<b>Client side reserved SAPs</b>	
No-station	0x00
Client Management Process	0x01
Public Client	0x10
<i>Open for client AP assignment</i>	0x02 ... 0x0F 0x11 and up
<b>Server side reserved SAPs</b>	
No-station	0x00
Management Logical Device	0x01
Reserved	0x02..0xF
<i>Open for server SAP assignment</i>	0x10...0x7E
All-station (Broadcast)	0x7F

## 10.8 LPWAN profile

### 10.8.1 Scope

This part of the DLMS UA Green Book specifies the DLMS/COSEM communication profile for Low-Power Wide Area Networks (LPWAN).

It specifies how the COSEM data model and the DLMS/COSEM application layer can be used over various LPWAN technologies using the IETF RFC 8724 “SCHC: Generic Framework for Static Context Header Compression and Fragmentation”, and in particular over LoRaWAN.

The DLMS/COSEM LPWAN communication profiles use connection-less transport layer based on the Internet Standard User Datagram Protocol (UDP) and Internet Protocol (IPv6).

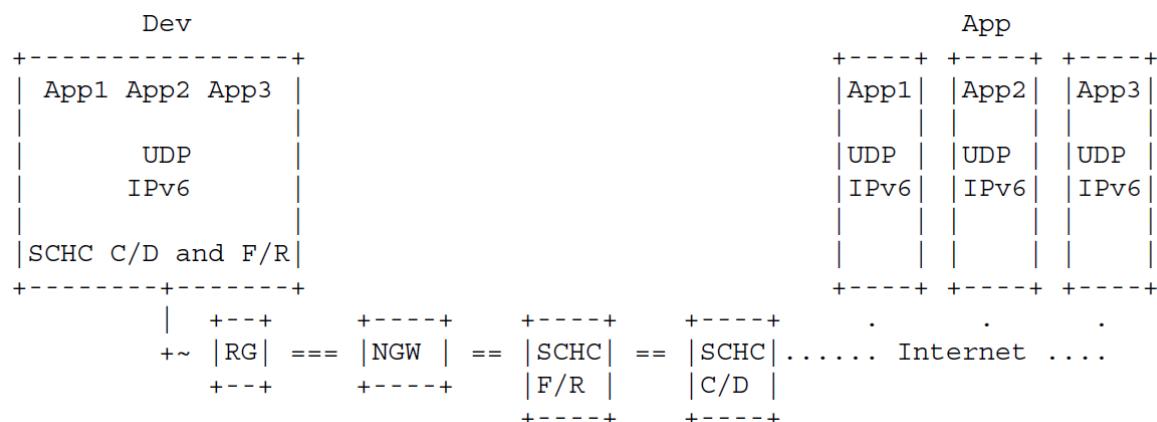
The adaptation layer is based on IETF RFC 8724, “SCHC: Generic Framework for Static Context Header Compression and Fragmentation” which provides both a header compression/decompression mechanism and an optional fragmentation/reassembly mechanism. SCHC compression is based on static context with small context identifier to represent full IPv6 / UDP / COSEM wrapper headers. If required, SCHC fragmentation is used to support IPv6 MTU over the LPWAN technologies.

### 10.8.2 Targeted communications environments

The DLMS/COSEM communication profiles for LPWAN networks are intended for remote data exchange on WAN between the HES and the end devices. The functional smart metering reference architecture is shown in Figure 158. From a DLMS point of view, they are connected directly to the HES via the G interface. All other interfaces and elements are out of the scope for this profile.

End devices comprise application functions and communication functions. They may be utility meters or any other kind of IoT Devices. They use UDP/IPv6, SCHC Compression / Decompression and Fragmentation / Reassembly features as specified in RFC 8724 and communicate with their related Application Server via the Network Gateway.

This profile maps to Figure 194:



**Figure 194 – LPWAN (SCHC) architecture outline**

### 10.8.3 Security

LPWAN technologies provide various lower layer security features. The application security features provided by DLMS/COSEM can be used over any of them.

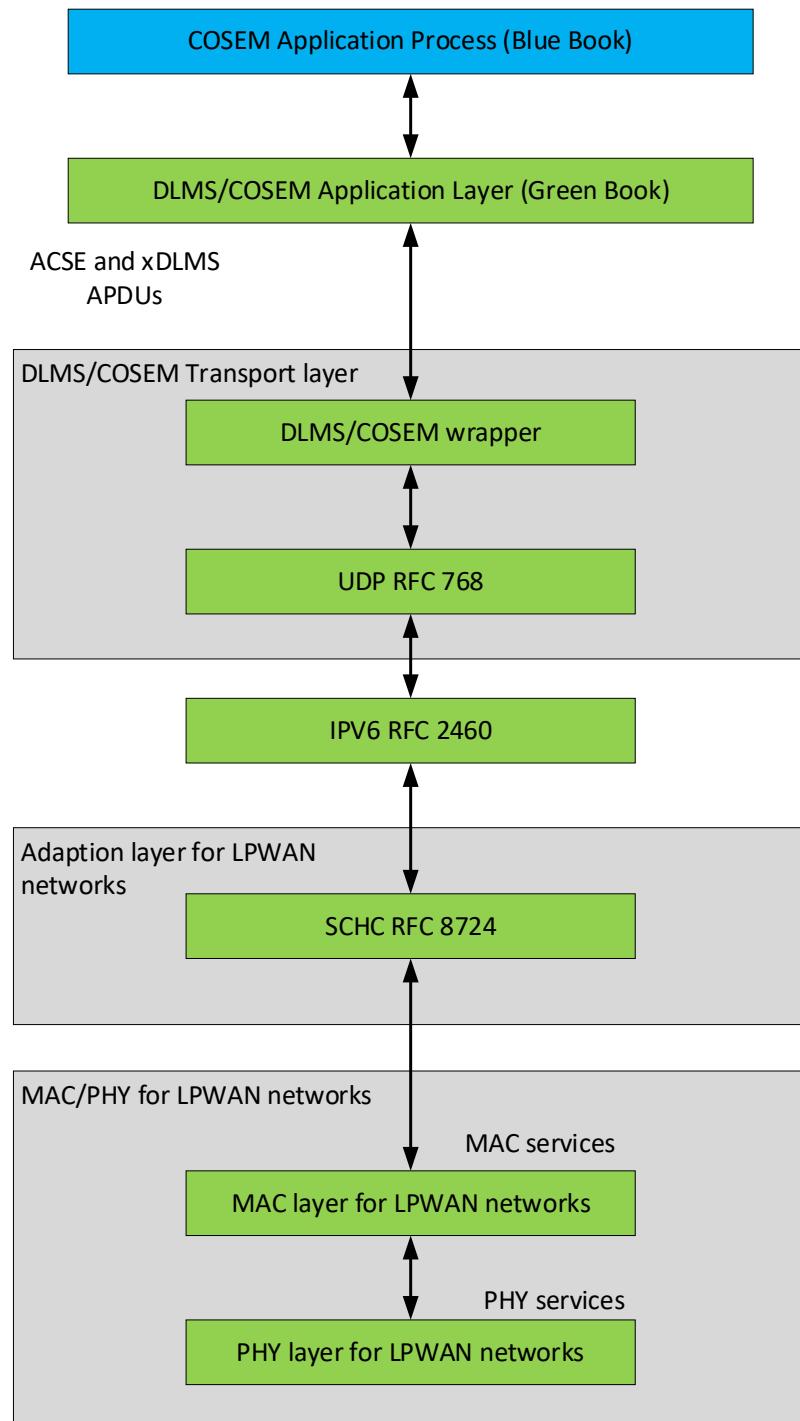
### 10.8.4 Use of the communications layers for this profile

#### 10.8.4.1 Information related to the use of the standard specifying the lower layers

IETF RFC 8724 can be considered as an adaptation layer between UDP/IPv6 and the underlying LPWAN technology. SCHC comprises two sublayers, compression and fragmentation that are independent of the specific LPWAN technology. RFC 8724 supports UDP/IPv6 and as such supports the DLMS/COSEM UDP/IP

Profile. No adaptations or limitations to RFC 8724 or DLMS/COSEM are expected to be required when using SCHC to transport COSEM APDUs.

### **10.8.5 Structure of the communication profiles**



**Figure 195 – The DLMS/COSEM LPWAN communication profile**

### **10.8.6 Lower protocol layers and their use**

### **10.8.6.1 Overview**

Lower layers are any LPWAN lower layers that can transport SCHC packets as specified in RFC 8724.

NOTE RFC 8376 provides an overview of LPWAN technologies that can be used for running IP in LPWANs.

ABP is not permitted when LoRaWAN network is being applied. See also DLMS UA 1000-1 Part 2 Ed.15:2021, 4.16.

#### **10.8.6.2 Physical layer**

Physical layer is out of scope of this documents, it is specific to the LPWAN technology used.

#### **10.8.6.3 MAC layer**

MAC layer is out of scope of this documents, it is specific to the LPWAN technology used.

#### **10.8.6.4 Adaptation layer**

The adaptation layer is compliant RFC 8724. It interfaces at the upper layer IPv6 as specified in RFC 2460.

#### **10.8.6.5 Service mapping and adaptation layers**

The DLMS/COSEM transport layer for IP networks performs the necessary binding of the COSEM object model and the DLMS/COSEM application layer in one part and the communication lower layers in the other part. The service mapping is fully specified in the UDP-DATA service, 7.2.3.2.2.

#### **10.8.6.6 Registration and connection management**

Registration and connection management are specific to each LPWAN technology and network. Details may be found in DLMS UA 1000-1 Part 2 Ed.15:2021, 4.16.

#### **10.8.7 Identification and addressing schemes**

The identification and addressing of SAPs is as described in Table 120.

**Table 120 – Client and server SAPs**

<b>Client SAPs</b>	
Client Management Process	0x01
Public Client	0x10
Open for client SAP assignment	0x02 ... 0x0F 0x11... 0xFF
<b>Server SAPs</b>	
Management Logical Device	0x01
Reserved for future use	0x02...0x0F
Open for server SAP assignment	0x10...0x7E
All-station (Broadcast)	0xFF

#### **10.8.8 Specific considerations for the application layer service**

##### **10.8.8.1 Overview**

The constraints and options available to AL services are those dictated by any UDP-based DLMS approach.

##### **10.8.8.2 Application Association establishment and release: ACSE services**

According to subclauses 10.3.6.1 and 10.3.6.2.

##### **10.8.8.3 xDLMS services**

According to subclauses 10.3.6.4 and 10.3.6.5.

#### 10.8.8.4 Security mechanisms

##### 10.8.8.4.1 DLMS/COSEM security

DLMS/COSEM security applies at the application layer and model level. As a consequence, application security does not depend on the structure of this communication profile. All the security mechanisms as defined in the DLMS UA Green Book and Blue Book are applicable without any restrictions. The security suites and the security policies chosen and the PKI to use are project specific. They depend on the project specific companion specification.

##### 10.8.8.4.2 Lower layers security

In addition to the DLMS/COSEM security, the lower protocol layers can also provide security features addressing confidentiality, data authenticity and integrity. These security features are out of the scope of this technical report.

##### 10.8.8.4.3 Registration and deregistration of lower layers security

Registration and deregistration security are specific to the LPWAN technology used (see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.16).

#### 10.8.8.5 Transferring long application messages

For transporting long messages, either the service specific block transfer or the general block transfer (GBT) DLMS/COSEM application layer mechanisms can be used (the latter is preferred).

#### 10.8.8.6 Media access, bandwidth and timing consideration

Out of scope of this technical report; these aspects are specific to the LPWAN technology used (see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.16).

#### 10.8.9 Communication configuration and management

The parameters allowing the configuration of the adaptation layer and the LPWAN lower layers are mapped to attributes and methods of DLMS/COSEM interface classes.

- a setup and a diagnostic IC for SCHC-LPWAN are specified in DLMS UA 1000-1 Part 2 Ed.15:2021, 4.16;
- a setup and a diagnostic IC should be specified for each specific LPWAN technology. For LoRaWAN they are specified in DLMS UA 1000-1 Part 2 Ed.15:2021, 4.16.

#### 10.8.10 The COSEM application process

All the features defined in the DLMS UA Green Book and Blue Book are applicable without any restrictions.

## 10.9 Wi-SUN profile

### 10.9.1 General

The Wi-SUN FAN is designed to fulfil the requirements of a ubiquitous network, but has no applications specified to operate over it, so it is application agnostic. Similarly, DLMS does not specify a single set of lower layers to support the application layer and can be described as transport agnostic.

The Wi-SUN FAN specification covers layers 1-4 of the OSI stack and handles UDP packets and may handle TCP packets also. DLMS/COSEM already specifies a communication profile for operating over TCP or UDP on IPv4 or IPv6 networks and thus there is no need for a specific adaptation layer.

This profile defines how DLMS/COSEM operates over a Wi-SUN network and to provide COSEM set up and diagnostic interface classes needed for creating a communication context over a Wi-SUN network. It is designed to be used over lossy networks and for efficiency assumes the use of 6LowPAN and RPL.

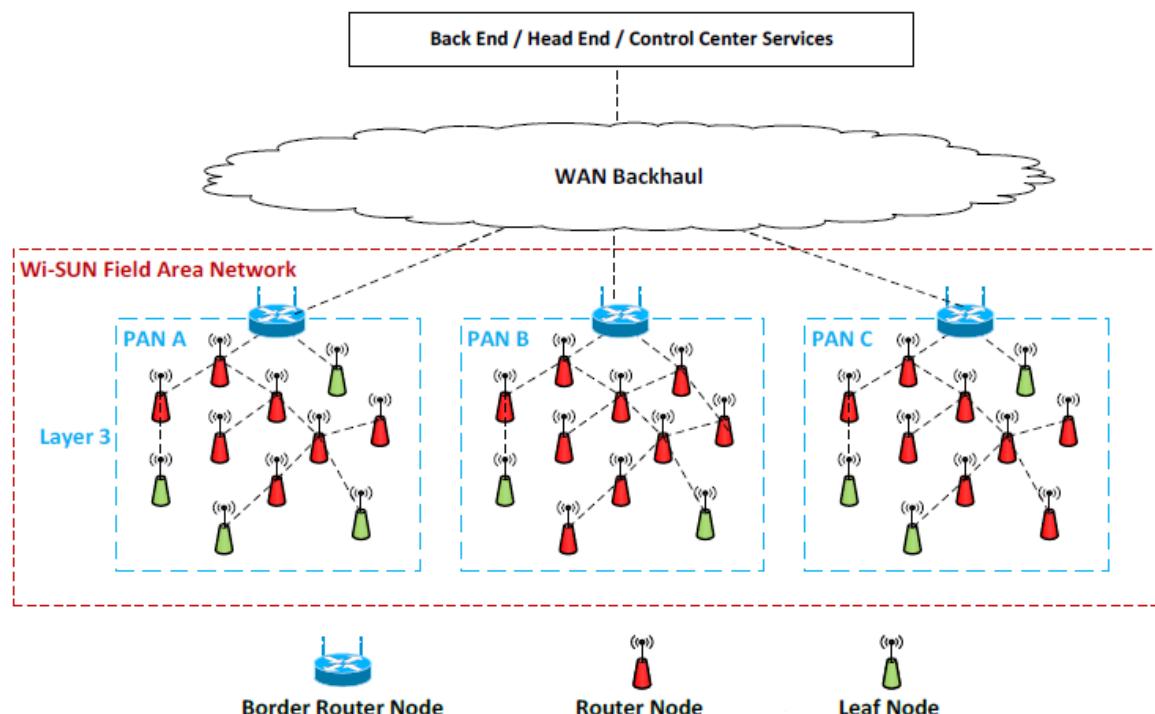
### 10.9.2 Profile description

#### 10.9.2.1 Targeted communication environments

As per Figure 158, this communication profile is concerned with interface “G”.

A Wi-SUN Field Area Network (FAN) is an IPv6 wireless mesh network based on IEEE 802.15.4, designed for critical infrastructure projects. Each Personal Area Network (PAN) is designed to support thousands of router devices with a single border router. A FAN may consist of many PANs allowing individual networks to scale to millions of devices, see Figure 196.

Border Router configuration is out of scope from the application layer perspective in this profile.



**Figure 196 – Wi-SUN Architecture (Layer 3 routing)**

### 10.9.3 Use of the communication layers for this communication profile

#### 10.9.3.1 Information related to the use of the standard specifying the lower layers

The Wi-SUN FAN is a mesh network described in the [FANSPEC] from the Wi-SUN Alliance. The [FANSPEC] is based on IEEE 802.15.4 and as such provides an interoperable profile or a kind of

companion specification to IEEE 802.15.4, specifying layers 1-4 of a communications stack terminating at an IPv6 transport layer.

The [FANSPEC] supports UDP/IP and as such supports the DLMS/COSEM UDP/IPv6 profile.

No adaptations or limitations to [FANSPEC] or to DLMS/COSEM are foreseen to be required in use of [FANSPEC] to transport DLMS/COSEM APDUs.

### 10.9.3.2 Structure of the communication profiles

Figure 197 shows the communication profile with a data viewpoint.

[FANSPEC] specifies two mechanisms for routing which may occur: RPL as per RFC 6550 (“route over”) and MHDS as per ANSI/TIA 4957.210 (“mesh under”). Routing using RPL shall be supported within the Wi-SUN specification and MHDS routing is optional. There are no special considerations for DLMS/COSEM regardless of which routing mechanism is implemented.

[FANSPEC] specifies the use of ICMPv6 as per RFC 4443 and RFC 6775 in the control plane.

[FANSPEC] specifies security mechanisms as shown in Figure 197. These have no impact on the security provided by DLMS/COSEM and it is-at the discretion of the implementer which security suites and security mechanisms to implement depending on the project requirements.

[FANSPEC] specifies nodes also support the MPL as described in RFC 7731.

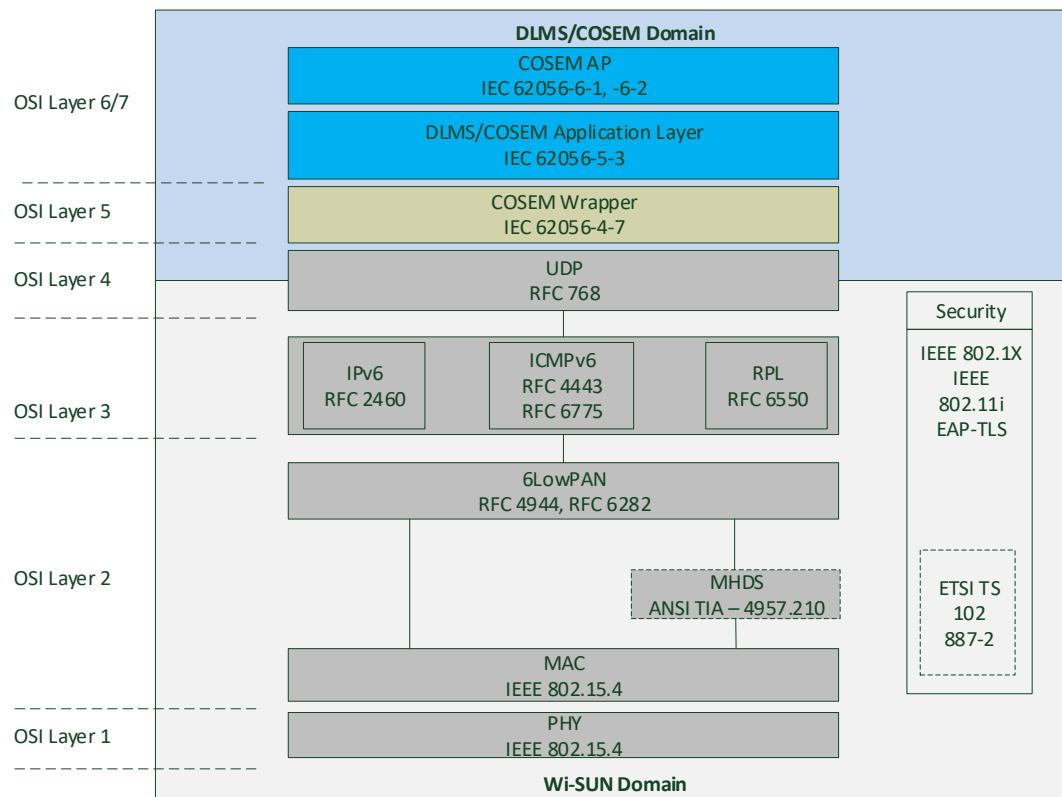


Figure 197 – Wi-SUN communication profile diagram

### 10.9.3.3 Lower protocol layers and their use

#### 10.9.3.3.1 Overview

The layers required for this communication profile up to the UDP layer are as specified in the [FANSPEC] and as shown in Figure 197 above.

#### 10.9.3.3.2 Physical layer

The PHY layer is defined as a sub-set of IEEE 802.15.4 as specified in [FANSPEC] and [PHYSPEC].

The parameters of the PHY layer may be specific to a particular regulatory domain. [PHYSPEC] Table 3 details the regulatory domain and operating class for each region. Together the regulatory domain value and operating class designate which frequency bands are supported, in line with [PHYSPEC], Table 3. To support this functionality, attributes of the same name have been included in the Wi-SUN set up IC and are described in attribute 11 channel\_plan (see DLMS UA 1000-1 Part 2 Ed.15:2021, 4.17.1).

**NOTE** The above sets the channel spacing, channel 0 frequency and number of channels plus the data rate.

The mandatory operating modes are specified in [PHYSPEC], 5.6.

#### 10.9.3.3.3 MAC layer

The MAC layer is defined as a sub-set of IEEE 802.15.4 as specified in [FANSPEC], 6.3.

#### 10.9.3.3.4 Network layer IPv6 parameters

[FANSPEC] prescribes the use of an IPv6 network layer and UDP Transport layer. The IPv6 addressing shall be as per [FANSPEC], 6.2.3.1.2.

[FANSPEC] specifies the use of IPv6 with 6LoWPAN adaptation, automated assignment of link-local IPv6 addresses, network layer routing and forwarding of unicast and multicast IPv6 packets.

The Network Service is provided by IPv6 as defined in RFC 2460 with 6LoWPAN adaptation as defined in RFC 4944 and RFC 6282, as per [FANSPEC], 5.4.1. ICMPv6, as defined in RFC 4443, is used for control plane information exchange.

As the [FANSPEC] includes Layer 3 routing (RPL) support as mandatory and optional Layer 2 multi-hop delivery service (MHDS), the 6LoWPAN Mesh Headers are not required and are not used.

Layer 3 routing shall be supported in accordance with RPL – see RFC 6550 – using the non-storing mode. IPv6 packets are routed hop-by-hop “upwards” toward the RPL root. IPv6 packets are source routed “downward” from the RPL root. As RFC 6550, 9.4 observes: “In Non-Storing mode, the root builds a strict source routing header, hop-by-hop, by recursively looking up one-hop information that ties a Target (address or prefix) and a transit address together.”

If the optional Layer 2 multihop delivery service is present, Layer 3 routing is not used and a Layer 2 multi-hop forwarding path to the destination is determined for the Layer 2 frame.

As per [FANSPEC], support for RPL is mandatory. However Multi-Hop Delivery Service (MHDS) can be operated as an alternative to RPL. The Wi-SUN setup IC – see DLMS UA 1000-1 Part 2 Ed.15:2021 – provides an attribute (attribute 3, routing\_method) that informs at the application process which mechanism is used.

**NOTE** There is no COSEM IC available to model MHDS in this version of the Specification.

#### 10.9.3.3.5 Constraints

The network layer for the FAN network has the following characteristics:

- the network layer is based on IPv6 with 6LoWPAN;
- network layer routing and forwarding of unicast IPv6 packets is supported;
- network layer routing and forwarding of multicast IPv6 packets is supported;
- automated assignment of link-local IPv6 addresses is supported.

#### 10.9.3.3.6 6LowPAN

FAN nodes support transmission and reception of uncompressed IPv6 packets over IEEE 802.15.4 networks as described in RFC 4944 and updated by RFC 6282 as per [FANSPEC], 6.2.3.2.1.

#### 10.9.3.3.7 Transport layer parameters

FAN nodes must implement the User Datagram Protocol as defined in RFC 768 and may also implement TCP as defined in RFC 793. No modifications to the protocols specified within these RFCs are required for use over Wi-SUN networks. The focus of this profile will be the UDP transport.

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	487/614
-----------------------	------------	-----------------------	---------

#### 10.9.3.4 Service mapping and adaptation layers

FAN nodes support the transmission and reception of IPv6 packets as per RFC 4944 and updated by RFC 6282. Further information is available in [FANSPEC], 6.2.3.1.1.

This profile relies on and is aligned with the existing DLMS/COSEM mechanisms for transporting DLMS/COSEM APDUs over IP Networks via UDP transport, using the wrapper described in 7.2.3.3 and the TCP-UDP/IP profile as described in 10.3.

As Wi-SUN implements a standard UDP or optionally TCP transport layer no further adaptation mechanism is required to transport DLMS/COSEM APDUs.

Figure 198 below shows the service primitive mapping between DLMS/COSEM and the Wi-SUN layers, which are consistent regardless of the routing method (layer 3 using RPL or layer 2 using MHDS). As network management is out of scope in this profile apart from some diagnostic information no other services are needed to lower layers. The server's application process shall provide all diagnostic information from lower layers to populate the diagnostic ICs.

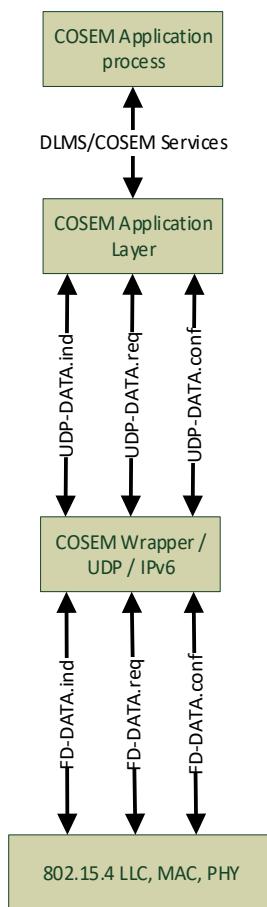


Figure 198 – Service mapping

#### 10.9.3.5 Registration and connection management

The lower layer registration and connection process within Wi-SUN occurs transparently to the application layer and will appear exactly as any other IP connection. For informative purposes and to give some context to the attributes described in the Wi-SUN setup IC, this clause describes what preconfigured parameters are needed to set up a Wi-SUN network.

This subclause describes attributes needed for the joining processes, as described in the [FANSPEC], 6.3.4.6.3.

In case of a discrepancy between this clause and the information in [FANSPEC] then [FANSPEC] shall take precedence.

Prior to deployment, a FAN node is administratively configured with the parameters shown in Table 121 and Security material including:

- a) Manufacturer IDevID;
- b) Private key corresponding to the IDevID;
- c) The remaining Manufacturing certificate chain from which was issued the IDevID, including the Manufacturer Root certificate;
- d) See [FANSPEC], 6.5.1 for definitions and [FANSPEC], 6.5.5 for recommendations regarding secure handling of the certificates and private key.

**Table 121 – FANSPEC to Wi-SUN setup IC attribute mapping**

FANSPEC Term	Wi-SUN setup interface class attribute
NETNAME-IE	network_name
Routing method	routing_method
Channel Plan	channel_plan
Regulatory Domain	channel_plan.regulatory_domain_identifier
Manufacturer IdevID	No equivalent DLMS/COSEM attribute and different from PAN ID
Trickle Timer Imin	default_dio_interval_min
Trickle Timer Imax	default_dio_interval_max calculated from default_dio_interval_min and default_dio_inetrvl_doublings
Join state	join_state

For Imin and Imax values for the discovery Trickle timers, see DLMS UA 1000-1 Part 2 Ed.15:2021,Wi-SUN setup (class\_id = 95).

The server shall be aware of its join state at any point in time. Join state has one of the following values:

**Table 122 – Join states**

Enum	meaning
0	Initialisation state
1	(Select PAN)
2	(Authenticate)
3	(Acquire PAN Config)
4	(Configure Routing)
5	(Operational)

The DLMS/COSEM application process on a device should only send traffic when the device is in join state 5. For further information see [FANSPEC], Figure 6-44.

This state shall be reported at the Application layer in attribute 14 of an instance of the Wi-SUN setup IC (class\_id = 95).

#### 10.9.4 Identification and addressing schemes

The identification and addressing of SAPs is as described in Table 1 and Table 2.

Table 123 summarises the recommended port numbers for use within this profile. The port numbers have been chosen to allow for the use of the 6LowPAN compression algorithm used within Wi-SUN.

**Table 123 – UDP port numbering**

<b>Application</b>	<b>DLMS Server UDP ports</b>		<b>DLMS Client UDP ports</b>	
	Source	Destination	Source	Destination
DLMS/COSEM	61616	61617 – 61629	61617 – 61629	61616

FAN nodes implement DHCPv6 – see RFC 3315 – to provide automated management of IPv6 GUA and ULA address assignments. Assigned IP addresses are as per [FANSPEC], 6.2.3.1.2.1.2.

GUA/ULA's assigned to the device will be stored in attribute 4 of the IPv6 setup IC instance dedicated to Wi-SUN. DHCPv6 is used here for 2 reasons:

- to control the FAN address pool;
- to provide delivery of additional configuration via DHCPv6 options.

Rapid Commit, using Solicit/Reply message exchange is the only DHCP operating mode used by the FAN. FAN nodes should not generate and may ignore receipt of Advertise, Rebind, Request, Renew, Release, Decline, Confirm, Reconfigure and Information messages.

#### 10.9.5 Specific considerations for the application layer services

##### 10.9.5.1 General

The constraints applicable and options available to AL services are those dictated by any UDP-based DLMS approach.

##### 10.9.5.2 Application Association establishment and release: ACSE services

The ACSE services may be used in any way that is applicable to UDP connections as described in 9.4.4.

Confirmed, unconfirmed and pre-established Application Associations may be used according to 9.4.4.

Persistent Application Associations may be used and are described in IEC 62056-8-20:2016, subclause 7.2.2.

##### 10.9.5.3 xDLMS services

Any of the xDLMS services may be used and must respect any restrictions placed by the use of UDP.

##### 10.9.5.4 Security mechanisms

There are no perceived restrictions to the use of DLMS/COSEM security.

The DLMS/COSEM application should not manage digital certificates for Wi-SUN networks.

Additional security is provided by [FANSPEC], 5.7 as reproduced below and as per Figure 197.

[FANSPEC] supports the following set of security mechanisms:

- 1) Layer 2 network access control, mutual authentication, and establishment of a secure pairwise link between a FAN node and its PAN Border Router is implemented with an adaptation of IEEE 802.1X and RFC 5216 for IEEE 802.15.4:
  - a) Joining nodes assume the role of Supplicant (SUP), Border Routers assume the role of Authenticator, and a AAA service (such as RADIUS) fulfills the role of the Authentication Service;
  - b) An EAPOL relay mechanism is defined to provide EAPOL messaging between Supplicant and Authenticator when the two are not within radio range of each other;
  - c) Mutual authentication is based on IEEE 802.1ar secure device identity, with specific requirements made for the format of the FAN IDevID (aka the device certificate);
  - d) The FAN supports both Wi-SUN approved vendor specific PKIs and a Wi-SUN PKI (provided for those vendors not desiring to operate a Certificate Authority as per [FANSPEC], 6.5.1);
- 2) An adaptation of the IEEE 802.11i 4-Way handshake establishes a shorter-lived secure pairwise link between the joining node and the PAN Border Router; the 4-Way handshake also establishes the PAN group security between all nodes of the PAN;
- 3) Secure node to node (pairwise) links are supported between one-hop FAN neighbors using an adaptation of ETSI-TS-102-887-2;
- 4) FAN nodes implement frame Security as specified in IEEE 802.15.4, 9, using either the group or node to node pairwise keys previously established;
- 5) Misbehaving nodes are removed from a PAN by the Border Router revoking the node's pairwise keys, followed by distribution of new group keys to the remaining PAN nodes.

#### **10.9.5.5 Transferring long application messages**

The usual DLMS approaches – General Block Transfer (GBT) and/or service specific block transfer – to transport long messages may be applied without restriction.

#### **10.9.5.6 Media access, bandwidth and timing considerations**

The Media Access Control (MAC) layer used in Wi-SUN is as described in IEEE 802.15.4. There are four frame exchange patterns defined:

- unicast Directed (DFE);
- unicast Extended Directed (EDFE);
- broadcast (BFE); and
- asynchronous, used for the discovery / join process.

FAN nodes operate in non-beacon mode.

Unslotted CSMA-CA and CCA mode 1 is supported as per IEEE 802.15.4, 6.2.5.1 and IEEE 802.15.4, 10.2.7 respectively.

As per [FANSPEC], the following channel access mechanisms are employed:

- 1) Asynchronous frame transmissions:
  - a) CSMA-CA is not used before asynchronous frame transmissions;
  - b) CCA mode 1 may be used before asynchronous frame transmissions.
- 2) DFE frame transmissions:
  - a) CSMA-CA is used with CCA mode 1 before DFE ULAD and EAPOL frame transmissions;
  - b) CSMA-CA and/or CCA is not used before Acknowledgement frame transmissions;
  - c) The minimum inter-frame spacing between 2 successive DFE frames by a node transmitted on the same channel is as described in IEEE 802.15.4.
- 3) EDFE frame transmissions:
  - a) CSMA-CA is not used;

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	491/614
-----------------------	------------	-----------------------	---------

- b) CCA mode 1 is used before initial frame transmissions and initial frame transmissions should begin near the start of the targeted receiver's dwell interval (taking into account clock drift);
  - c) CCA is not used before response frames.
- 4) Broadcast frame transmissions:
- a) CSMA-CA and CCA mode 1 are used.

#### **10.9.6 Communication configuration and management**

For all aspects visible to the COSEM application process, any required existing configuration COSEM interface classes specified in DLMS UA 1000-1 Part 2 Ed.15:2021 shall be referenced.

See DLMS UA 1000-1 Part 2 Ed.15:2021 for the interface classes to be applied.

#### **10.9.7 The COSEM application process**

There are no special requirements for the COSEM application process.

## 10.10 Gateway protocol

### 10.10.1 General

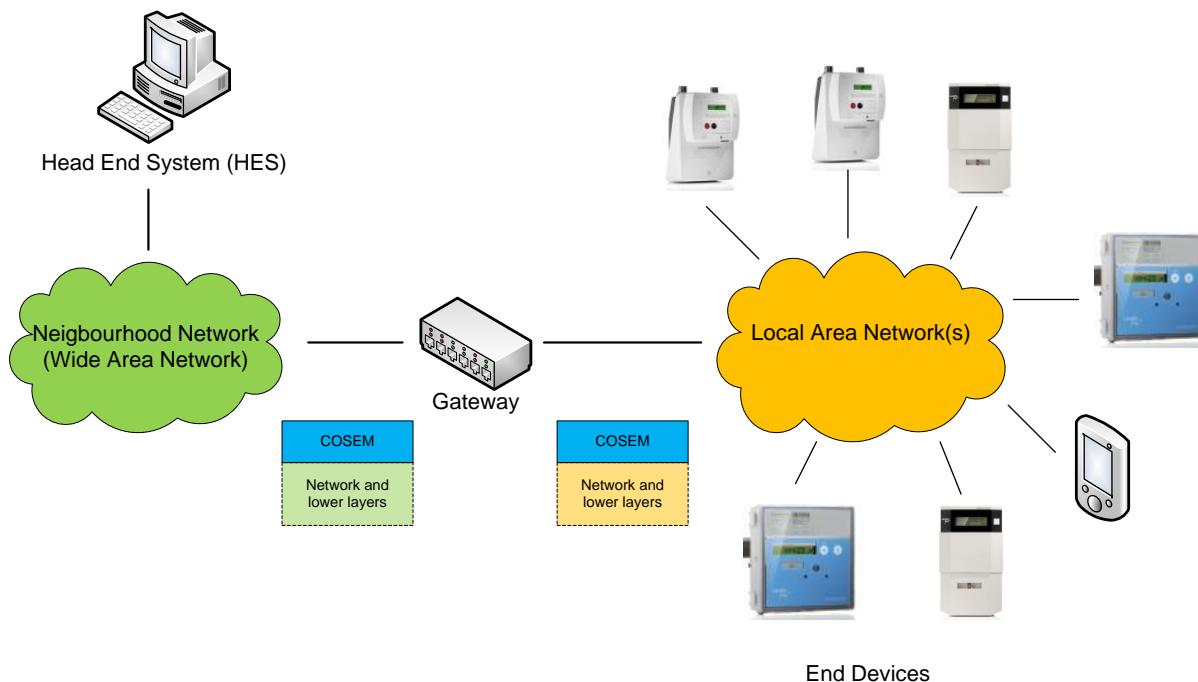
This subclause specifies a method for exchanging data between DLMS clients and servers via a gateway, when this gateway is connected to a Wide Area Network (WAN) or to a Neighbourhood Network (NN) on the one hand, and to a Local Network (LAN) on the other hand, with DLMS servers connected to this LAN.

The gateway acts bidirectional, i.e. it is also possible for a server in the LAN to send messages to a client in the WAN/NN using the gateway (push application).

The gateway function itself may be implemented in a DLMS/COSEM device or in a stand-alone device.

The DLMS/COSEM specification for **device** data exchange is based on the client/server model, where the head end system (HES) acts as a client requesting services, and the end devices (e.g. meters) act as servers providing the services requested. In many cases, the client can reach each **device** directly, using unicast, multicast or broadcast messages.

There are cases however, when it is practical to connect several end devices to a LAN, and reach those devices via a gateway. The protocol stack used on the LAN may be the same as the one used on the WAN/NN or it may be different.



**Figure 199 – General architecture with gateway**

The DLMS client (HES) reaches the gateway via a WAN or via NNAP; see Figure 199. The gateway itself may be a stand-alone device or a DLMS/COSEM device capable of acting as a gateway. If configured accordingly, it passes the COSEM APDUs transparently between the HES or NNAP and the COSEM servers (end devices).

The gateway may act bidirectional, i.e. it is also possible for a COSEM server (end device) in the LAN to send COSEM APDUs to the COSEM client (HES) in the WAN/NN using the gateway (push application).

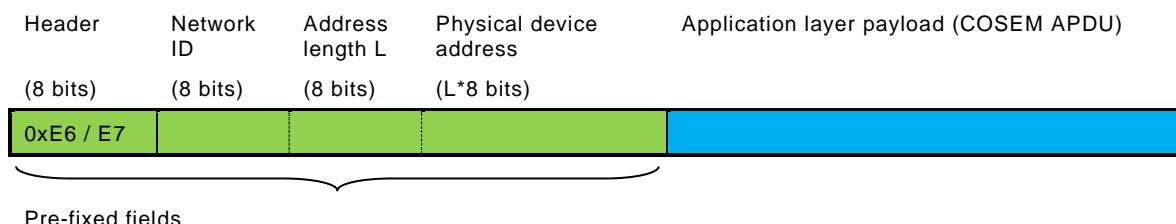
### 10.10.2 The gateway protocol

The top layer of any DLMS/COSEM communication profile is the DLMS/COSEM application layer.

In order to leave both, the suite of lower layers and the DLMS/COSEM AL unaffected, the task of routing each message from the client to the end device on the LAN is solved by pre-fixing the COSEM APDUs with a few bytes specifying the network to be used and the address of the device on the LAN and vice versa.

The gateway extracts the payload, a COSEM APDU – together with the application addresses – from the WAN/NN protocol and puts it as a payload to the LAN protocol, and the other way round.

The structure of the prefix with four fields is shown in Figure 200.



**Figure 200 – The fields used for pre-fixing the COSEM APDUs**

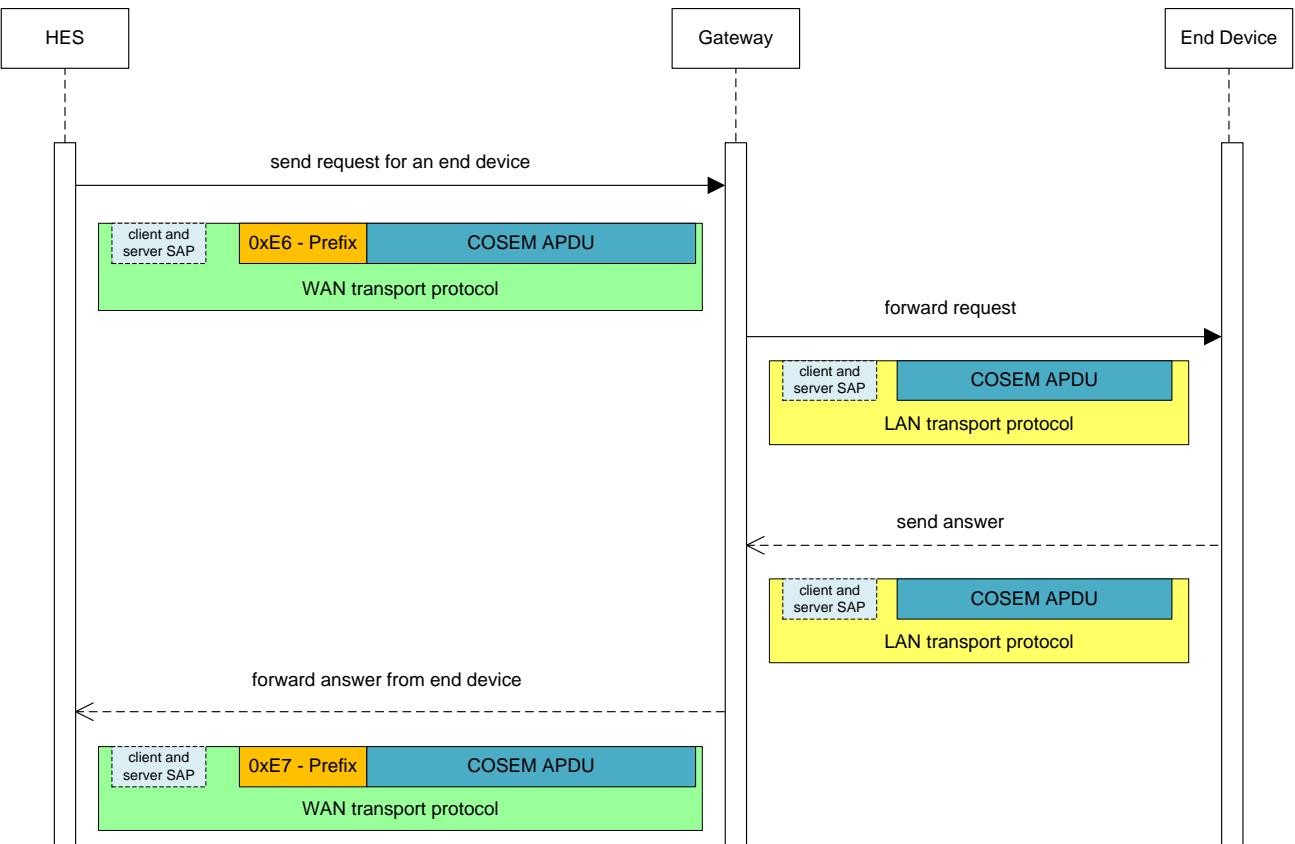
- the value of the first byte (header) is either 0xE6 or 0xE7. It indicates that the following bytes don't contain a plain COSEM APDU but contain a COSEM APDU with a prefix:
  - 0xE6 indicates a request message from a DLMS client to a DLMS server or a request message from a DLMS server to a DLMS client (data notification);
  - 0xE7 indicates a response from the DLMS server to the DLMS client;
- the second byte carries an identifier of the destination network where the messages are transferred to. This allows accessing several networks using the same or different communication protocols through the same gateway. The network ID is not linked to the communication protocol and can be set to any value. If only one network exists, 0x00 shall be used.
- the third byte defines the length of the physical device address given in the next L bytes. It depends on the communication protocol used.
- bytes 4 to 4+(L-1) carry the physical device address of the end device or the HES as requested by the communication protocol.

When a telegram with pre-fixed fields reaches a device, which is not a gateway or which doesn't support pre-fixed fields, it shall be simply discarded.

When the client exchanges data directly with the master **device**, the pre-fixed fields are not present.

### 10.10.3 HES in the WAN/NN acting as Initiator (Pull operation)

In the sequence diagram shown in Figure 201 the traditional pull data exchange between the DLMS client and server via a gateway is further elaborated:

**Figure 201 – Pull message sequence chart**

**Prerequisite:** The client in the WAN/NN has to know the network ID, the protocol and the physical device address of the server it wants to reach in the LAN.

The DLMS client (HES) sends every request, carried by a COSEM APDU, prefixed with four fields as shown in Figure 200 using the protocol layer supporting the DLMS/COSEM AL on the WAN/NN.

The gateway forwards each COSEM APDU carrying a .request service primitive to the appropriate network using the network ID and the physical device address contained in the pre-fixed fields. The client and server SAPs are extracted from the protocol layer supporting the DLMS/COSEM AL on the WAN/NN and inserted into the supporting protocol layer of the DLMS/COSEM AL on the LAN.

The APDUs carrying the requests do not have any prefix when they arrive in the end devices in the LAN. Every end device processes the request and provides the answer the same way as if it's connected directly to the client.

When the device responds to a request, it is done as if it's connected to the client directly: The APDU does not need to be pre-fixed.

When the gateway receives a COSEM APDU carrying a .response service primitive on the LAN, it extracts the client and server SAPs from the protocol layer supporting the DLMS/COSEM AL on the LAN. Afterwards it inserts them into the supporting protocol layer of the DLMS/COSEM AL on the WAN/NN and sends the message with pre-fixed fields to the client using the WAN/NN protocol.

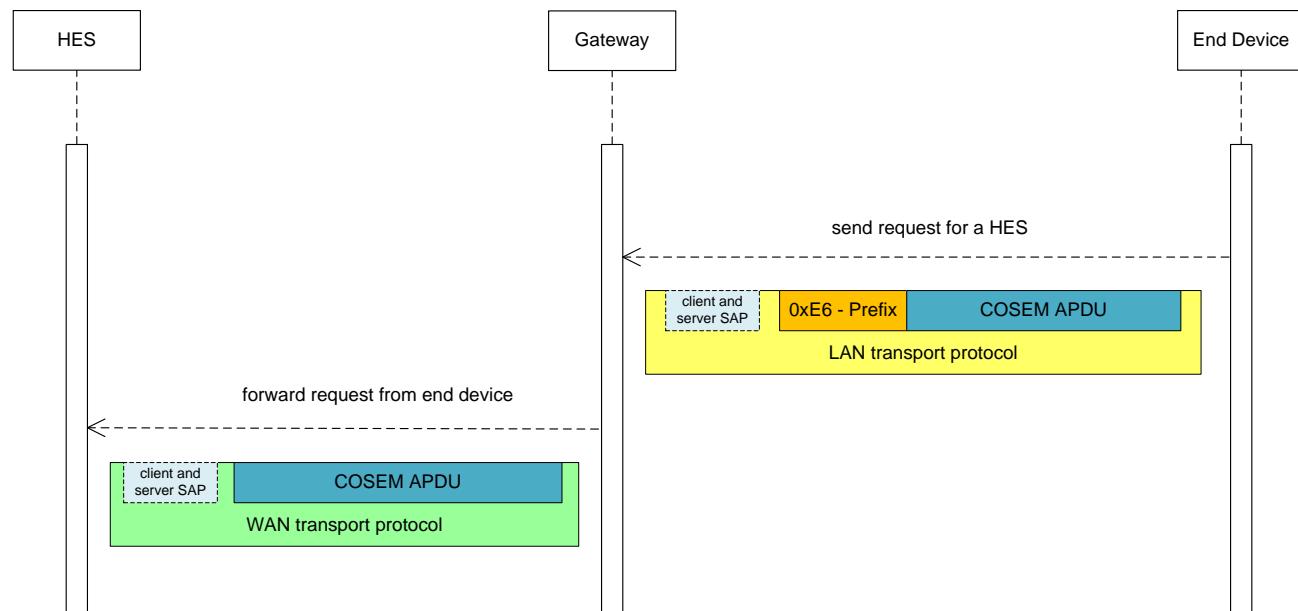
#### 10.10.4 End devices in the LAN acting as Initiators (Push operation)

##### 10.10.4.1 General

It is also possible for a server (end device) in the LAN to send messages to a client (HES) in the WAN / NN using the gateway without having received a request service before (push application). Depending on the capabilities of the gateway two scenarios are supported.

#### 10.10.4.2 End device with WAN/NN knowledge

Precondition: The server in the LAN has to know the network ID, the protocol and the address of the client it wants to reach in the WAN/NN.



**Figure 202 – Push message sequence chart**

The server (end device) sends every request (e.g. data notification request), carried by the COSEM APDU, pre-fixed with 4 fields as defined before to the gateway using the protocol layer supporting the DLMS/COSEM AL on the LAN, as shown in Figure 202.

The gateway forwards each COSEM APDU carrying a .request service primitive using the network ID, the protocol and the client address (e.g. WAN/NN MAC address) contained in the pre-fixed fields.

The client and server SAPs are extracted from the protocol layer supporting the DLMS/COSEM AL on the LAN and inserted into the supporting protocol layer of the DLMS/COSEM AL on the WAN/NN.

#### 10.10.4.3 End devices without WAN/NN knowledge

If the end device has no knowledge on the WAN/NN network or if it has no knowledge if it is connected to a gateway at all, it can send standard (not pre-fixed) data notification requests to the gateway. It is then the duty of the gateway to further deal with such messages.

Since this does not require a protocol extension, this use case is not described any further.

#### 10.10.5 Security

The DLMS/COSEM AL security mechanisms ensure end-to-end security through the gateway.

## 11 AARQ and AARE encoding examples

### 11.1 General

This Clause contains examples of encoding the AARQ and AARE APDUs, in cases of using various levels of authentication and in cases of success and failure.

The AARQ, AARE, RLRQ and RLRE APDUs – see GB 9.4.3.1 – shall be encoded in BER (**ISO/IEC 8825-1:2015**). The user-information field of the AARQ and AARE APDUs contains the xDLMS InitiateRequest / InitiateResponse or confirmedServiceError APDUs respectively, encoded in A-XDR as OCTET STRING.

### 11.2 Encoding of the xDLMS InitiateRequest / InitiateResponse APDU

The xDLMS InitiateRequest / InitiateResponse APDUs are specified as follows:

```

InitiateRequest ::= SEQUENCE
{
  -- shall not be encoded in DLMS without ciphering
  dedicated-key                               OCTET STRING OPTIONAL,
  response-allowed                           BOOLEAN DEFAULT TRUE,
  proposed-quality-of-service                 IMPLICIT Integer8 OPTIONAL,
  proposed-dlms-version-number               Unsigned8,
  proposed-conformance                      Conformance,
  client-max-receive-pdu-size                Unsigned16
}

InitiateResponse ::= SEQUENCE
{
  negotiated-quality-of-service             IMPLICIT Integer8 OPTIONAL,
  negotiated-dlms-version-number           Unsigned8,
  negotiated-conformance                  Conformance,
  server-max-receive-pdu-size              Unsigned16,
  vaa-name                                ObjectName
}

```

The xDLMS InitiateRequest and InitiateResponse APDUs are encoded in A-XDR and they are inserted in the user-information field of the AARQ / AARE APDU respectively.

In the examples below, the following values are used:

- dedicated key: not present; no ciphering is used;
- response-allowed: TRUE (default value);
- proposed-quality-of-service and negotiated-quality-of-service: not present (not used in DLMS/COSEM);
- proposed-conformance and negotiated-conformance: see below;
- proposed-dlms-version-number and negotiated-dlms-version-number = 6;
- client-max-receive-pdu-size:  $1200_D = 0x04B0$ ;
- server-max-receive-pdu-size:  $500_D = 0x01F4$ ;
- vaa-name in the case of LN referencing: the dummy value  $0x0007$ ;
- vaa-name in the case of SN referencing: the base\_name of the current Association SN object,  $0xFA00$ .
- The proposed-conformance and the negotiated-conformance elements carry the proposed conformance block and the negotiated conformance block respectively. The values of these examples, for LN referencing and SN referencing respectively, are shown in Table 124.

**Table 124 – Conformance block**

Conformance ::= [APPLICATION 31] IMPLICIT BIT STRING (SIZE(24))	Used with	LN referencing		SN referencing	
		Proposed	Negotiated	Proposed	Negotiated
-- the bit is set when the corresponding service or functionality is available					
reserved-zero (0),		0	0	0	0
reserved-one (1),		0	0	0	0
reserved-two (2),		0	0	0	0
read (3),	SN	0	0	1	1
write (4),	SN	0	0	1	1
unconfirmed-write (5),	SN	0	0	1	1
reserved-six (6),		0	0	0	0
reserved-seven (7),		0	0	0	0
attribute0-supported-with-set (8),	LN	0	0	0	0
priority-mgmt-supported (9),	LN	1	1	0	0
attribute0-supported-with-get (10),	LN	1	0	0	0
block-transfer-with-get-or-read (11),	LN	1	1	0	0
block-transfer-with-set-or-write (12),	LN	1	0	0	0
block-transfer-with-action (13),	LN	1	0	0	0
multiple-references (14),	LN / SN	1	0	1	1
information-report (15),	SN	0	0	1	1
reserved-sixteen (16),		0	0	0	0
reserved-seventeen (17),		0	0	0	0
parameterized-access (18),	SN	0	0	1	1
get (19),	LN	1	1	0	0
set (20),	LN	1	1	0	0
selective-access (21),	LN	1	1	0	0
event-notification (22),	LN	1	1	0	0
action (23)	LN	1	1	0	0
Value of the bit string		00 7E 1F	00 50 1F	1C 03 20	1C 03 20

With these parameters, the A-XDR encoding of the xDLMS InitiateRequest APDU is as shown in Table 125:

**Table 125 – A-XDR encoding the xDLMS InitiateRequest APDU**

<b>-- A-XDR encoding the xDLMS InitiateRequest APDU</b>	<b>LN referencing</b>	<b>SN referencing</b>
// encoding of the tag of the xDLMS APDU CHOICE <i>(InitiateRequest)</i>	01	01
-- encoding of the dedicated-key component ( <b>OCTET STRING OPTIONAL</b> )		
// usage flag ( <b>FALSE</b> , not present)	00	00
-- encoding of the response-allowed component ( <b>BOOLEAN DEFAULT TRUE</b> )		
// usage flag ( <b>FALSE</b> , default value <b>TRUE</b> conveyed)	00	00
-- encoding of the proposed-quality-of-service component ([0] <b>IMPLICIT Integer8 OPTIONAL</b> )		
// usage flag ( <b>FALSE</b> , not present)	00	00
-- encoding of the proposed-dlms-version-number component ( <b>Unsigned8</b> )		
// value= 6, the encoding of an Unsigned8 is its value	06	06
-- encoding of the proposed-conformance component (Conformance, [APPLICATION 31] <b>IMPLICIT BIT STRING</b> (SIZE(24))) <sup>1</sup>		
// encoding of the [APPLICATION 31] tag (ASN.1 explicit tag) <sup>2</sup>	5F 1F	5F 1F
// encoding of the length of the 'contents' field in octet (4)	04	04
// encoding of the number of unused bits in the final octet of the BIT STRING (0)	00	00
// encoding of the fixed length BIT STRING value	00 7E 1F	1C 03 20
-- encoding of the client-max-receive-pdu-size component ( <b>Unsigned16</b> )		
// value = 0x04B0, the encoding of an Unsigned16 is its value	04 B0	04 B0
-- resulting octet-string, to be inserted in the user-information field of the AARQ APDU	01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0	01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0

<sup>1</sup> As specified in IEC 61334-6, Annex C, Examples 1 and 2, the proposed-conformance element of the xDLMS InitiateRequest APDU and the negotiated-conformance element of the xDLMS InitiateResponse APDU are encoded in BER. That's why the length of the bit-string and the number of the unused bits are encoded.

<sup>2</sup> For encoding of identifier octets, see ISO/IEC 8825-1:2015, 8.1.2. For compliance with existing implementations, encoding of the [Application 31] tag on one byte (5F) instead of two bytes (5F 1F) is accepted when the 3-layer, connection-oriented, HDLC based profile is used.

The A-XDR encoding of the xDLMS InitiateResponse APDU is as shown in Table 126.

**Table 126 – A-XDR encoding the xDLMS InitiateResponse APDU**

-- A-XDR encoding the xDLMS InitiateResponse APDU	LN referencing	SN referencing
// encoding of the tag of the xDLMS APDU CHOICE (InitiateResponse)	08	08
-- encoding of the negotiated-quality-of-service component ([0] <b>IMPLICIT Integer8 OPTIONAL</b> )		
// usage flag( <b>FALSE</b> , not present)	00	00
-- encoding of the negotiated-dlms-version-number component (Unsigned8)		
// value = 6, the encoding of an Unsigned8 is its value	06	06
-- encoding of the negotiated-conformance component (Conformance, [APPLICATION 31] <b>IMPLICIT BIT STRING</b> (SIZE(24)))		
// encoding of the [APPLICATION 31] tag (ASN.1 explicit tag)	5F 1F	5F 1F
// encoding of the length of the 'contents' field in octet (4)	04	04
// encoding of the number of unused bits in the final octet of the BIT STRING (0)	00	00
// encoding of the fixed length BIT STRING value	00 50 1F	1C 03 20
-- encoding of the server-max-receive-pdu-size component (Unsigned16)		
// value = 0x01F4, the encoding of an Unsigned16 is its value	01 F4	01 F4
-- encoding of the VAA-Name component (ObjectName, Integer16)		
// value=0x0007 for LN and 0xFA00 for SN referencing; the encoding of a value constrained Integer16 is its value	00 07	FA 00
-- resulting octet-string, to be inserted in the user-information field of the AARE APDU	08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07	08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00

### 11.3 Specification of the AARQ and AARE APDU

The AARQ and the AARE APDUs are specified in 9.5 as follows:

```

AARQ-apdu ::= [APPLICATION 0] IMPLICIT SEQUENCE
{
  -- [APPLICATION 0] == [ 60H ] = [ 96 ]
    protocol-version           [0] IMPLICIT BIT STRING {version1 (0)} DEFAULT
{version1},
    application-context-name   [1]          Application-context-name,
    called-AP-title            [2]          AP-title OPTIONAL,
    called-AE-qualifier        [3]          AE-qualifier OPTIONAL,
    called-AP-invocation-id   [4]          AP-invocation-identifier OPTIONAL,
    called-AE-invocation-id   [5]          AE-invocation-identifier OPTIONAL,
    calling-AP-title            [6]          AP-title OPTIONAL,
    calling-AE-qualifier       [7]          AE-qualifier OPTIONAL,
    calling-AP-invocation-id   [8]          AP-invocation-identifier OPTIONAL,
    calling-AE-invocation-id   [9]          AE-invocation-identifier OPTIONAL,
-- The following field shall not be present if only the kernel is used.
  sender-acse-requirements   [10] IMPLICIT ACSE-requirements OPTIONAL,
-- The following field shall only be present if the authentication functional unit is selected.
  mechanism-name             [11] IMPLICIT Mechanism-name OPTIONAL,
-- The following field shall only be present if the authentication functional unit is selected.
  calling-authentication-value [12] EXPLICIT Authentication-value OPTIONAL,
}

```

```

implementation-information [29] IMPLICIT Implementation-data OPTIONAL,
user-information [30] EXPLICIT Association-information OPTIONAL
}

-- The user-information field shall carry an InitiateRequest APDU encoded in A-XDR, and then
-- encoding the resulting OCTET STRING in BER.

AARE-apdu ::= [APPLICATION 1] IMPLICIT SEQUENCE
{
  -- [APPLICATION 1] == [ 61H ] = [ 97 ]

  protocol-version [0] IMPLICIT BIT STRING {version1 (0)} DEFAULT
{version1},
    application-context-name [1] Application-context-name,
    result [2] Association-result,
    result-source-diagnostic [3] Associate-source-diagnostic,
    responding-AP-title [4] AP-title OPTIONAL,
    responding-AE-qualifier [5] AE-qualifier OPTIONAL,
    responding-AP-invocation-id [6] AP-invocation-identifier OPTIONAL,
    responding-AE-invocation-id [7] AE-invocation-identifier OPTIONAL,

  -- The following field shall not be present if only the kernel is used.
  responder-acse-requirements [8] IMPLICIT ACSE-requirements OPTIONAL,

  -- The following field shall only be present if the authentication functional unit is selected.
  mechanism-name [9] IMPLICIT Mechanism-name OPTIONAL,

  -- The following field shall only be present if the authentication functional unit is selected.
  responding-authentication-value [10] EXPLICIT Authentication-value OPTIONAL,
  implementation-information [29] IMPLICIT Implementation-data OPTIONAL,
  user-information [30] EXPLICIT Association-information OPTIONAL
}

-- The user-information field shall carry either an InitiateResponse (or, when the proposed
-- xDLMS context is not accepted by the server, a confirmedServiceError APDU encoded in
-- A-XDR, and then encoding the resulting OCTET STRING in BER.

```

## 11.4 Data for the examples

In these examples:

- the protocol-version is the default ACSE version;
- the value of the application-context-name:
  - in the case of LN referencing, with no ciphering: 2, 16, 756, 5, 8, 1, 1;
  - in the case of SN referencing, with no ciphering: 2, 16, 756, 5, 8, 1, 2;
- the optional called-AP-title, called-AE-qualifier, called-AP-invocation-id, called-AE-invocation-id, calling-AP-title, calling-AE-qualifier, calling-AP-invocation-id, calling-AE-invocation-id fields of the AARQ, and the optional responding-AP-title, responding-AE-qualifier, responding-AP-invocation-id, responding-AE-invocation-id, of the AARE are not present;
- the value of the mechanism-name:
  - in the case of low-level-security: 2, 16, 756, 5, 8, 2, 1;
  - in the case of high-level-security (5): 2, 16, 756, 5, 8, 2, 5;
- the calling-authentication-value:
  - in the case of low-level-security is 12345678 (encoded as 31 32 33 34 35 36 37 38);
  - in the case of high-level security, (challenge CtoS) is K56iVagY (encoded as 4B 35 36 69 56 61 67 59);
- the responding authentication-value (challenge StoC) is P6wRJ21F (encoded as 50 36 77 52 4A 32 31 46);
- the optional implementation-information field in the AARQ and AARE APDUs is not present;
- the user-information field carries the xDLMS InitiateRequest / InitiateResponse APDUs as shown above.

The application-context-name and the (authentication) mechanism-name OBJECT IDENTIFIERS are encoded as follows:

- BER Encoding for OBJECT IDENTIFIER is a packed sequence of numbers representing the arc labels. Each number – except the first two, which are combined into one – is represented as a

- series of octets, with 7 bits being used from each octet and the most significant bit is set to 1 in all but the last octet. The fewest possible number of octets must be used;
- in the case of the application context name LN referencing with no ciphering the arc labels of the object identifier are (2, 16, 756, 5, 8, 1, 1);
    - the first octet of the encoding is the combination of the first two numbers into a single number, following the rule of  $40 \cdot \text{First} + \text{Second} \rightarrow 40 \cdot 2 + 16 = 96 = 0x60$ ;
    - the third number of the Object Identifier (756) requires two octets: its hexadecimal value is 0x02F4, which is 00000010 11110100, but following the above rule, the MSB of the first octet shall be set to 1 and the MSB of the second (last) octet shall be set to 0, thus this bit shall be shifted into the LSB of the first octet. This gives binary 10000101 01110100, which is 0x8574;
    - each remaining numbers of the Object Identifier required to be encoded on one octet;
    - this results in the encoding 60 85 74 05 08 01 01.
  - similarly, in the case of application context name SN referencing with no ciphering the BER encoding is 60 85 74 05 08 01 02;
  - in the case of mechanism name low-level-security, the BER encoding is 60 85 74 05 08 02 01;
  - in the case of mechanism name high-level-security (5), the BER encoding is 60 85 74 05 08 02 05.

### 11.5 Encoding of the AARQ APDU

Here, six different cases are shown:

- LN referencing with no ciphering, no security, LLS and HLS;
- SN referencing with no ciphering, no security, LLS and HLS;

The encoding is shown in Table 127. See also Table 128.

**Table 127 – BER encoding the AARQ APDU**

-- BER encoding the AARQ APDU	LN referencing			SN referencing		
	no sec.	LLS	HLS	no sec.	LLS	HLS
// encoding of the tag of the AARQ APDU ([APPLICATION 0], Application)	60			60		
// encoding of the length of the AARQ's content's field	1D	36	36	1D	36	36
-- protocol-version field ([0], <b>IMPLICIT BIT STRING</b> { version1 (0) } <b>DEFAULT</b> { version1 })						
// no encoding, thus it is considered with its <b>DEFAULT</b> value						
-- encoding the fields of the Kernel						
-- application-context-name field ([1], Application-context-name, <b>OBJECT IDENTIFIER</b> )						
// encoding of the tag ([1], Context-specific)	A1			A1		
// encoding of the length of the tagged component's value field	09			09		
// encoding of the choice for application-context-name ( <b>OBJECT IDENTIFIER</b> , Universal)	06			06		
// encoding of the length of the Object Identifier's value field	07			07		
// encoding of the value of the Object Identifier	60 85 74 05 08 01 01			60 85 74 05 08 01 02		
-- encoding the fields of the authentication functional unit						
--sender-acse-requirements field ([10], ACSE-requirements, <b>BIT STRING</b> { authentication (0) } )						
// encoding of the tag of the acse-requirements field ([10], <b>IMPLICIT</b> , Context-specific)	-	8A	8A	-	8A	8A
// encoding of the length of the tagged component's value field	-	02	02	-	02	02
// encoding of the number of unused bits in the last byte of the <b>BIT STRING</b>	-	07	07	-	07	07
// encoding of the authentication functional unit (0) NOTE The number of bits coded may vary from client to client, but within the COSEM environment, only bit 0 set to 1 (indicating the requirement of the authentication functional unit) is to be respected.	-	80	80	-	80	80
-- mechanism-name field ([11], <b>IMPLICIT</b> Mechanism-name <b>OBJECT IDENTIFIER</b> )						

-- BER encoding the AARQ APDU	LN referencing			SN referencing		
	no sec.	LLS	HLS	no sec.	LLS	HLS
// encoding of the tag ([11], <b>IMPLICIT</b> , Context-specific)	-	8B	8B	-	8B	8B
// encoding of the length of the tagged component's value field	-	07	07	-	07	07
// encoding of the value of the <b>OBJECT IDENTIFIER</b> :						
- low-level-security-mechanism-name (1),	-	60 85 74 05 08 02 01	60 85 74 05 08 02 05	-	60 85 74 05 08 02 01	60 85 74 05 08 02 05
- high-level-security-mechanism-name (5)						
-- calling-authentication-value field ([12], Authentication-value <b>CHOICE</b> )						
// encoding of the tag ([12], <b>EXPLICIT</b> , Context-specific)	-	AC	AC	-	AC	AC
// encoding of the length of the tagged component's value field	-	0A	0A	-	0A	0A
// encoding of the choice for Authentication-value (charstring [0] <b>IMPLICIT GraphicString</b> )	-	80	80	-	80	80
// encoding of the length of the Authentication-value's value field (8 octets)	-	08	08	-	08	08
// encoding the calling-authentication-value:						
- in the case of LLS, the value of the Password "12345678"	-	31 32 33 34 35 36 37 38	4B 35 36 69 56 61 67 59	-	31 32 33 34 35 36 37 38	4B 35 36 69 56 61 67 59
- in the case of HLS, the value of challenge CtoS "K56ivagY"						
-- encoding the user-information field component (Association-information, <b>OCTET STRING</b> )						
// encoding the tag ([30], Context-specific, Constructed)	BE	BE	BE	BE	BE	BE
// encoding of the length of the tagged component's value field	10	10	10	10	10	10
// encoding the choice for user-information ( <b>OCTET STRING</b> , Universal)	04	04	04	04	04	04
// encoding of the length of the <b>OCTET STRING</b> 's value field (14 octets)	0E	0E	0E	0E	0E	0E
// user-information: xDLMS InitiateRequest APDU	01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0			01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0		

**Table 128 – The complete AARQ APDU**

LN referencing with no ciphering, lowest level security	60 1D A1 09 06 07 60 85 74 05 08 01 01 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0
LN referencing with no ciphering, low level security	60 36 A1 09 06 07 60 85 74 05 08 01 01 8A 02 07 80 8B 07 60 85 74 05 08 02 01 AC 0A 80 08 31 32 33 34 35 36 37 38 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0
LN referencing with no ciphering, high level security	60 36 A1 09 06 07 60 85 74 05 08 01 01 8A 02 07 80 8B 07 60 85 74 05 08 02 05 AC 0A 80 08 4B 35 36 69 56 61 67 59 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0
SN referencing with no ciphering, lowest level security	60 1D A1 09 06 07 60 85 74 05 08 01 02 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0
SN referencing with no ciphering, low level security	60 36 A1 09 06 07 60 85 74 05 08 01 02 8A 02 07 80 8B 07 60 85 74 05 08 02 01 AC 0A 80 08 31 32 33 34 35 36 37 38 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0
SN referencing with no ciphering, high level security	60 36 A1 09 06 07 60 85 74 05 08 01 02 8A 02 07 80 8B 07 60 85 74 05 08 02 05 AC 0A 80 08 4B 35 36 69 56 61 67 59 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0

## 11.6 Encoding of the AARE APDU

Here, six different cases are shown:

- LN referencing with no ciphering, no security or LLS, successful establishment of the AA;
- LN referencing with no ciphering, no security or LLS, failure because the proposed application-context-name does not fit the application-context-name supported by the server (failure case 1):
  - when the device uses LN referencing, SN referencing is proposed;
  - when the device uses SN referencing, LN referencing is proposed;
- LN referencing with no ciphering, no security or LLS, failure because the proposed-dlms-version-number is too low; (failure case 2)
- LN referencing with no ciphering, HLS, successful establishment of the AA;
- SN referencing with no ciphering, no security or LLS, successful establishment of the AA;
- SN referencing with no ciphering, HLS, successful establishment of the AA;

The encoding is shown in Table 129. See also Table 130.

**Table 129 – BER encoding the AARE APDU**

-- BER encoding the AARE APDU	LN referencing				SN referencing	
	No sec./LLS success	No sec./LLS failure 1	No sec./LLS failure 2	HLS success	No sec./LLS success	HLS success
// encoding of the tag for the AARE APDU ([APPLICATION 1], Application)	61				61	
// encoding of the length of the AARE's content's field	29	29	1F	42	29	42
-- protocol-version field ([0], <b>IMPLICIT BIT STRING</b> { version1 (0) } <b>DEFAULT</b> { version1 })						
// no encoding, thus it is considered with its <b>DEFAULT</b> value						
-- application-context-name field ([1], Application-context-name, <b>OBJECT IDENTIFIER</b> )						
// encoding of the tag ([1], Context-specific)	A1					A1
// encoding of the length of the tagged component's value field	09					09
// encoding the choice for application-context-name ( <b>OBJECT IDENTIFIER</b> , Universal)	06					06
// encoding of the length of the Object Identifier's value field	07					07
// encoding of the value of the Object Identifier: NOTE when the proposed application-context does not fit the application-context supported by the server, the server may respond with the application-context name proposed or the application-context-name supported.	60 85 74 05 08 01 01	60 85 74 05 08 01 01	60 85 74 05 08 01 01 or 60 85 74 05 08 01 02	60 85 74 05 08 01 01	60 85 74 05 08 01 02	60 85 74 05 08 01 02
-- result field ([2], Association-result, <b>INTEGER</b> )						
// encoding of the tag ([2], Context-specific)	A2					A2
// encoding of the length of the tagged component's value field	03					03
// encoding of the choice for the result ( <b>INTEGER</b> , Universal)	02					02
// encoding of the length of the result's value field	01					01

-- BER encoding the AARE APDU	LN referencing				SN referencing	
	No sec./LLS success	No sec./LLS failure 1	No sec./LLS failure 2	HLS success	No sec./LLS success	HLS success
// encoding of the value of the Result: // success: 0, accepted // failure case 1 and 2: 1, rejected-permanent	00	01	01	00	00	00
-- result-source-diagnostic field ([3], Associate-source-diagnostic, <b>CHOICE</b> )						
// encoding of the tag ([3], Context-specific)	A3				A3	
// encoding of the length of the tagged component's value field	05				05	
// encoding of the tag for the acse-service-user CHOICE (1)	A1				A1	
// encoding of the length of the tagged component's value field	03				03	
// encoding of the choice for associate-source-diagnostics ( <b>INTEGER</b> , Universal)	02				02	
// encoding of the length of the value field	01				01	
// encoding of the value: - success, no security and LLS: 0, no diagnostics provided; - failure 1: 2, application-context-name not supported; - failure 2: 1, no-reason-given. - success, HLS security (5): 14, authentication required;	00	02	01	0E	00	0E
-- encoding the fields of the authentication functional unit						
-- responder-acse-requirements field ([8], <b>IMPLICIT</b> , ACSE-requirements, <b>BIT STRING</b> { authentication (0) } )						
// encoding of the tag of the acse-requirements field ([8], <b>IMPLICIT</b> , Context-specific)	-			88	-	88
// encoding of the length of the tagged component's value field.	-			02	-	02

-- BER encoding the AARE APDU	LN referencing				SN referencing	
	No sec./LLS success	No sec./LLS failure 1	No sec./LLS failure 2	HLS success	No sec./LLS success	HLS success
// encoding of the number of unused bits in the last byte of the <b>BIT STRING</b>	-			07	-	07
// encoding of the authentication functional unit (0)	-			80	-	80
-- mechanism-name field ([9], <b>IMPLICIT</b> , Mechanism-name <b>OBJECT IDENTIFIER</b> )						
// encoding of the tag ([9], <b>IMPLICIT</b> , Context-specific)	-			89	-	89
// encoding of the length of the tagged component's value field	-			07	-	07
// encoding the value of the object identifier: - high-level-security-mechanism-name (5)	-			60 85 74 05 08 02 05	-	60 85 74 05 08 02 05
-- responding-authentication-value field ([10], <b>EXPLICIT</b> , Authentication-value CHOICE)						
// encoding of the tag ([10], Context-specific)	-	-	--	AA	-	AA
// encoding of the length of the tagged component's value field	-	-	-	0A	-	0A
// encoding of the choice for Authentication-value (charstring [0] <b>IMPLICIT</b> GraphicString)	-	-	-	80	-	80
// encoding of the length of the Authentication-information's value field (8 octets)	-	-	-	08	-	08
// encoding of the value of the challenge Stoc "P6wRJ21F"	-	-	-	50 36 77 52 4A 32 31 46	-	50 36 77 52 4A 32 31 46
-- encoding the user-information field component (Association-information, <b>OCTET STRING</b> )						
// encoding the tag for the user-information field component ([30], Context-specific, Constructed)	BE	BE	BE	BE	BE	BE
// encoding of the length of the tagged component's value field	10	10	06	10	10	10
// encoding of the choice for user-information ( <b>OCTET STRING</b> , Universal)	04	04	04	04	04	04
// encoding of the length of the OCTET STRING's value field	0E	0E	04	0E	0E	0E

<b>-- BER encoding the AARE APDU</b>	<b>LN referencing</b>				<b>SN referencing</b>	
	<b>No sec./LLS success</b>	<b>No sec./LLS failure 1</b>	<b>No sec./LLS failure 2</b>	<b>HLS success</b>	<b>No sec./LLS success</b>	<b>HLS success</b>
// failure case 1: xDLMS InitiateResponse; // failure case 2: ConfirmedServiceError ([14]), InitiateError [1], ServiceError, initiate [6], dlms- version-too-low (1))	08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07	08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07	0E 01 06 01	08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07	08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00	08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00

**Table 130 – The complete AARE APDU**

LN referencing with no ciphering, no security or LLS, successful establishment of the AA	61 29 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07
LN referencing with no ciphering, no security or LLS, failure because the proposed application-context-name does not fit the application-context-name supported by the server (failure case 1)	61 29 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 01 A3 05 A1 03 02 01 02 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07 or 61 29 A1 09 06 07 60 85 74 05 08 01 02 A2 03 02 01 01 A3 05 A1 03 02 01 02 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07
LN referencing with no ciphering, no security or LLS, failure because the proposed-dlms-version-number is too low; (failure case 2)	61 1F A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 01 A3 05 A1 03 02 01 01 BE 06 04 04 0E 01 06 01
LN referencing with no ciphering, high level security	61 42 A1 09 06 07 60 85 74 05 08 01 01 A2 03 02 01 00 A3 05 A1 03 02 01 0E 88 02 07 80 89 07 60 85 74 05 08 02 05 AA 0A 80 08 50 36 77 52 4A 32 31 46 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07
SN referencing with no ciphering, lowest level security	61 29 A1 09 06 07 60 85 74 05 08 01 02 A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 10 04 0E 08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00
SN referencing with no ciphering, high level security	61 42 A1 09 06 07 60 85 74 05 08 01 02 A2 03 02 01 00 A3 05 A1 03 02 01 0E 88 02 07 80 89 07 60 85 74 05 08 02 05 AA 0A 80 08 50 36 77 52 4A 32 31 46 BE 10 04 0E 08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00

## 12 Encoding examples: AARQ and AARE APDUs using a ciphered application context

### 12.1 A-XDR encoding of the xDLMS InitiateRequest APDU, carrying a dedicated key

NOTE The System Title is the same in each example. In reality, the System Title in the request and in the response APDUs should be different, as they are originated by different systems.

In this example:

- the value of the dedicated key is 00112233445566778899AABBCCDDEEFF;
- the value of the Conformance block is 007E1F;
- the value of the client-max-receive-pdu-size is 1 200 bytes (0x04B0).

The A-XDR encoding of the xDLMS InitiateRequest APDU carrying a dedicated key is shown in Table 131.

**Table 131 – A-XDR encoding of the xDLMS InitiateRequest APDU**

// encoding of the tag of the xDLMS APDU CHOICE (InitiateRequest)	01
-- encoding of the dedicated-key component ( <b>OCTET STRING OPTIONAL</b> )	
// usage flag ( <b>TRUE</b> , present)	01
// length of the <b>OCTET STRING</b>	10
// contents of the <b>OCTET STRING</b>	0011223344556677 8899AABBCCDDEEFF
-- encoding of the response-allowed component ( <b>BOOLEAN DEFAULT TRUE</b> )	
// usage flag ( <b>FALSE</b> , default value <b>TRUE</b> conveyed)	00
-- encoding of the proposed-quality-of-service component ([0] <b>IMPLICIT Integer8 OPTIONAL</b> )	
// usage flag ( <b>FALSE</b> , not present)	00
-- encoding of the proposed-dlms-version-number component ( <b>Unsigned8</b> )	
// value = 6; the A-XDR encoding of an Unsigned8 is its value	06
-- encoding of the proposed-conformance component ( <b>Conformance, [APPLICATION 31] IMPLICIT BIT STRING (SIZE(24))</b> ) <sup>1</sup>	
// encoding of the [APPLICATION 31] tag (ASN.1 explicit tag) <sup>2</sup>	5F1F
// encoding of the length of the 'contents' field in octet (4)	04
// encoding of the number of unused bits in the final octet of the BIT STRING (0)	00
// encoding of the fixed length BIT STRING value	007E1F
-- encoding of the client-max-receive-pdu-size component ( <b>Unsigned16</b> )	
// value = 0x04B0, the encoding of an Unsigned16 is its value	04B0
-- resulting octet-string	0101100011223344 5566778899AABBCC DDEEFF0000065F1F 0400007E1F04B0

<sup>1</sup> As specified in IEC 61334-6:2000 Annex C, Examples 1 and 2, the proposed-conformance element of the xDLMS InitiateRequest APDU and the negotiated-conformance element of the xDLMS InitiateResponse APDU are encoded in BER. That's why the length of the bit-string and the number of the unused bits are encoded.

<sup>2</sup> For encoding of identifier octets, see ITU-T X690, 8.1.2. For compliance with existing implementations, encoding of the [Application 31] tag on one byte (5F) instead of two bytes (5F 1F) is accepted when the 3-layer, connection-oriented, HDLC based profile is used.

## 12.2 Authenticated encryption of the xDLMS InitiateRequest APDU

Table 132 shows the encoding of an xDLMS InitiateRequest APDU which is also authenticated and encrypted using service-specific global ciphering.

**Table 132 – Authenticated encryption of the xDLMS InitiateRequest APDU using service-specific global ciphering**

	<i>X</i>	Contents	<i>LEN(X) bytes</i>	<i>len(X) bits</i>
<b>Security material</b>				
Security suite		GCM-AES-128		
System Title	<i>Sys-T</i>	<b>4D4D4D0000BC614E</b> (here, the five last octets contain the manufacturing number in hexa)	8	64
Invocation Counter	<i>IC</i>	01234567	4	32
Initialization Vector	<i>IV</i>	<i>Sys-T</i>    <i>IC</i> 4D4D4D0000BC614E01234567	12	96
Block cipher key (global)	<i>EK</i>	000102030405060708090A0B0C0D0E0F	16	128
Authentication Key	<i>AK</i>	D0D1D2D3D4D5D6D7D8D9DADBDCCDDDEF	16	128
<b>Security applied</b>		<b>Authenticated encryption</b>		
Security control byte (with unicast key)	<i>SC</i>	<i>SC-AE</i> 30	1	8
Security header	<i>SH</i>	<i>SH</i> = <i>SC-AE</i>    <i>IC</i> 3001234567	5	40
<b>Inputs</b>				
xDLMS APDU to be protected	<i>APDU</i>	01011000112233445566778899AABBCC DDEEFF0000065F1F0400007E1F04B0	31	188
Plaintext	<i>P</i>	01011000112233445566778899AABBCC DDEEFF0000065F1F0400007E1F04B0	31	188
Associated data	<i>A</i>	<i>SC</i>    <i>AK</i> 30D0D1D2D3D4D5D6D7D8D9DADBDCCDDDEF	17	136
<b>Outputs</b>				
Ciphertext	<i>C</i>	801302FF8A7874133D414CED25B42534 D28DB0047720606B175BD52211BE68	31	188
Authentication tag	<i>T</i>	41DB204D39EE6FDB8E356855	12	96
The complete ciphered APDU		<i>TAG</i>    <i>LEN</i>    <i>SH</i>    <i>C</i>    <i>T</i> 21303001234567801302FF8A7874133D 414CED25B42534D28DB0047720606B17 5BD52211BE6841DB204D39EE6FDB8E35 6855	50	400

## 12.3 The AARQ APDU

In this example, the following values are used:

- Application-Context-Name: Logical\_Name\_Refencing\_With\_Ciphering;
- Calling-AP-Title (carries the System Title): 4D4D4D0000BC614E;
- Mechanism-Name: COSEM\_low\_level\_security;
- Calling-Authentication-Value: 12345678.

The BER encoding of the AARQ APDU is shown in Table 133.

**Table 133 – BER encoding of the AARQ APDU**

// encoding of the tag of the AARQ APDU ([APPLICATION 0], Application)	60
// encoding of the length of the AARQ's contents field (102 octets)	66
-- protocol-version field ([0], <b>IMPLICIT BIT STRING</b> { version 1 (0) } <b>DEFAULT</b> { version1 })	
// no encoding, thus it is considered with its <b>DEFAULT</b> value	
-- encoding of the fields of the Kernel	
-- application-context-name field ([1], Application-context-name, <b>OBJECT IDENTIFIER</b> )	
// encoding of the tag ([1], Context-specific)	A1
// encoding of the length of the tagged component's value field	09
// encoding of the choice for application-context-name ( <b>OBJECT IDENTIFIER</b> , Universal)	06
// encoding of the length of the Object Identifier's value field	07
// encoding of the value of the Object Identifier	60857405080103
-- encoding of the calling-AP-title field	
// encoding of the tag [6], Context-specific)	A6
// encoding of the length of the tagged component's value field	0A
// encoding of the type ([4], Universal, Octetstring type)	04
// encoding of the length of the calling-AP-title-field	08
// encoding of the value	4D4D4D0000BC614E
-- encoding of the fields of the authentication functional unit	
--sender-acse-requirements field ([10], ACSE-requirements, <b>BIT STRING</b> { authentication (0) })	
// encoding of the tag of the acse-requirements field ([10], <b>IMPLICIT</b> , Context-specific)	8A
// encoding of the length of the tagged component's value field	02
// encoding of the number of unused bits in the last byte of the <b>BIT STRING</b>	07
// encoding of the authentication functional unit (0) NOTE The number of bits coded may vary from client to client, but within the COSEM environment, only bit 0 set to 1 (indicating the requirement of the authentication functional unit) is to be respected.	80
-- mechanism-name field ([11], <b>IMPLICIT</b> Mechanism-name <b>OBJECT IDENTIFIER</b> )	
// encoding of the tag [11], <b>IMPLICIT</b> , Context-specific)	8B
// encoding of the length of the tagged component's value field	07
// encoding of the value of the <b>OBJECT IDENTIFIER</b> : - low-level-security-mechanism-name,	60857405080201
-- calling-authentication-value field ([12], Authentication-value <b>CHOICE</b> )	
// encoding of the tag ([12], <b>EXPLICIT</b> , Context-specific)	AC
// encoding of the length of the tagged component's value field	0A
// encoding of the choice for Authentication-value (charstring [0] <b>IMPLICIT GraphicString</b> )	80
// encoding of the length of the Authentication-value's value field (8 octets)	08

// encoding the calling-authentication-value (12345678)	3132333435363738
-- encoding the user-information field component (Association-information, <b>OCTET STRING</b> )	
// encoding of the tag [30], Context-specific, Constructed)	BE
// encoding of the length of the tagged component's value field	34
// encoding of the choice for user-information ( <b>OCTET STRING</b> , Universal)	04
// encoding of the length of the <b>OCTET STRING</b> 's value field (32 octets)	32
-- ciphered xDLMS InitiateRequest APDU	2130300123456780 1302FF8A7874133D 414CED25B42534D2 8DB0047720606B17 5BD52211BE6841DB 204D39EE6FDB8E35 6855

## 12.4 A-XDR encoding of the xDLMS InitiateResponse APDU

In this example:

- the value of the Conformance block is 007C1F;
- the value of the server-max-receive-pdu-size is 1024 bytes (0x0400).

The A-XDR encoding of the xDLMS InitiateResponse APDU using service-specific global ciphering is shown in Table 134.

**Table 134 – A-XDR encoding of the xDLMS InitiateResponse APDU using service-specific global ciphering**

// encoding of the tag of the xDLMS APDU CHOICE (InitiateResponse)	08
-- encoding of the negotiated-quality-of-service component ([0] <b>IMPLICIT Integer8 OPTIONAL</b> )	
// usage flag ( <b>FALSE</b> , not present)	00
-- encoding of the negotiated-dlms-version-number component ( <b>Unsigned8</b> )	
// value = 6, the A-XDR encoding of an Unsigned8 is its value	06
-- encoding of the negotiated-conformance component (Conformance, [APPLICATION 31] <b>IMPLICIT BIT STRING (SIZE(24))</b> )	
// encoding of the [APPLICATION 31] tag (ASN.1 explicit tag)	5F1F
// encoding of the length of the 'contents' field in octet (4)	04
// encoding of the number of unused bits in the final octet of the BIT STRING (0)	00
// encoding of the fixed length BIT STRING value	007C1F
-- encoding of the server-max-receive-pdu-size component ( <b>Unsigned16</b> )	
// value = 0x0400, the encoding of an Unsigned16 is its value	0400
-- encoding of the VAA-Name component ( <b>ObjectName, Integer16</b> )	
// value=0x0007; the encoding of a value constrained Integer16 is its value	0007
-- resulting octet-string, to be inserted in the user-information field of the AARE APDU	0800065F1F040000 7C1F04000007

## 12.5 Authenticated encryption of the xDLMS InitiateResponse APDU

Table 135 shows the encoding of the xDLMS InitiateResponse APDU which is also authenticated and encrypted.

**Table 135 – Authenticated encryption of the xDLMS InitiateResponse APDU**

	<i>X</i>	Contents	<i>LEN(X)</i> <i>bytes</i>	<i>len(X)</i> <i>bits</i>
<b>Security material</b>				
Security suite		GCM-AES-128		
System Title	<i>Sys-T</i>	4D4D4D0000BC614E (here, the five last octets contain the manufacturing number in hexa)	8	64
Invocation Counter	<i>IC</i>	01234567	4	32
Initialization Vector	<i>IV</i>	<i>Sys-T</i>    <i>IC</i> 4D4D4D0000BC614E01234567	12	96
Block cipher key (global)	<i>EK</i>	000102030405060708090A0B0C0D0EOF	16	128
Authentication Key	<i>AK</i>	D0D1D2D3D4D5D6D7D8D9DADBDCCDDDEF	16	128
<b>Security applied</b>		<b>Authenticated encryption</b>		
Security control byte (with unicast key)	<i>SC</i>	<i>SC-AE</i> 30	1	8
Security header	<i>SH</i>	<i>SH</i> = <i>SC-AE</i>    <i>IC</i> 3001234567	5	40
<b>Inputs</b>				
xDLMS APDU to be protected	<i>APDU</i>	0800065F1F0400007C1F04000007	14	112
Plaintext	<i>P</i>	0800065F1F0400007C1F04000007	14	112
Associated data	<i>A</i>	<i>SC</i>    <i>AK</i> 30D0D1D2D3D4D5D6D7D8D9DADBDCCDDDEF	17	136
<b>Outputs</b>				
Ciphertext	<i>C</i>	891214A0845E475714383F65BC19	14	112
Authentication tag	<i>T</i>	745CA235906525E4F3E1C893	12	96
The complete Ciphered APDU		<i>TAG</i>    <i>LEN</i>    <i>SH</i>    <i>C</i>    <i>T</i> 281F3001234567891214A0845E475714 383F65BC19745CA235906525E4F3E1C8 93	33	264

## 12.6 The AARE APDU

The BER encoding of the AARE APDU is shown in Table 136.

**Table 136 – BER encoding of the AARE APDU**

-- BER encoding of the AARE APDU	
// encoding of the tag for the AARE APDU ([APPLICATION 1], Application)	61
// encoding of the length of the AARE's content's field (72 octets)	48
-- protocol-version field ([0], IMPLICIT BIT STRING { version1 (0) } DEFAULT { version1 })	
// no encoding, thus it is considered with its DEFAULT value	

-- BER encoding of the AARE APDU	
-- encoding of the fields of the Kernel	
-- application-context-name field ([1], Application-context-name, <b>OBJECT IDENTIFIER</b> )	
// encoding of the tag ([1], Context-specific)	A1
// encoding of the length of the tagged component's value field	09
// encoding of the choice for application-context-name ( <b>OBJECT IDENTIFIER</b> , Universal)	06
// encoding of the length of the Object Identifier's value field	07
// encoding of the value of the Object Identifier: NOTE when the proposed application-context does not fit the application-context supported by the server, the server may respond with the application-context name proposed or the application-context-name supported.	60857405080103
-- result field ([2], Association-result, <b>INTEGER</b> )	
// encoding of the tag ([2], Context-specific)	A2
// encoding of the length of the tagged component's value field	03
// encoding of the choice for the result ( <b>INTEGER</b> , Universal)	02
// encoding of the length of the result's value field	01
// encoding of the value of the Result: 0, accepted	00
-- result-source-diagnostic field ([3], Associate-source-diagnostic, <b>CHOICE</b> )	
// encoding of the tag ([3], Context-specific)	A3
// encoding of the length of the tagged component's value field	05
// encoding of the tag for the acse-service-user CHOICE (1)	A1
// encoding of the length of the tagged component's value field	03
// encoding of the choice for associate-source-diagnostics ( <b>INTEGER</b> , Universal)	02
// encoding of the length of the value field	01
// encoding of the value (0, no diagnostics provided)	00
-- encoding of the responding-AP-title field	
// encoding of the tag ([4], Context-specific)	A4
// encoding of the length of the tagged component's value field	0A
// encoding of the type ([4], Universal, Octetstring type)	04
// encoding of the length of the responding-AP-title-field	08
// encoding the value	4D4D4D0000BC614E
-- encoding of the fields of the authentication functional unit -- In this example the Authentication functional unit is not present; it is not necessary in the case of LLS, but if it is present it is also acceptable.	
-- encoding the user-information field component (Association-information, <b>OCTET STRING</b> )	
// encoding of the tag ([30], Context-specific, Constructed)	BE
// encoding of the length of the tagged component's value field	23
// encoding of the choice for user-information ( <b>OCTET STRING</b> , Universal)	04
// encoding of the length of the <b>OCTET STRING</b> 's value field	21

-- BER encoding of the AARE APDU	
-- ciphered xDLMS InitiateResponse APDU	281F300123456789 1214A0845E475714 383F65BC19745CA2 35906525E4F3E1C8 93

## 12.7 The RLRQ APDU (carrying a ciphered xDLMS InitiateRequest APDU)

The BER encoding of the RLRQ APDU is shown in Table 137.

**Table 137 – BER encoding of the RLRQ APDU**

-- BER encoding of the RLRQ APDU	
// encoding of the tag of the RLRQ APDU ([APPLICATION 2], Application)	62
// encoding of the length of the RLRQ's contents field	39
-- reason field	
// encoding of the tag ([0], IMPLICIT)	80
// encoding of the length of the tagged component's value field	01
// encoding of the value (0, normal)	00
-- encoding the user-information field component (Association-information, OCTET STRING)	
// encoding of the tag ([30], Context-specific, Constructed)	BE
// encoding of the length of the tagged component's value field	34
// encoding of the choice for user-information (OCTET STRING, Universal)	04
// encoding of the length of the OCTET STRING's value field (14 octets)	32
// user-information: xDLMS InitiateRequest APDU	2130300123456780 1302FF8A7874133D 414CED25B42534D2 8DB0047720606B17 5BD52211BE6841DB 204D39EE6FDB8E35 6855

## 12.8 The RLRE APDU (carrying a ciphered xDLMS InitiateResponse APDU)

The BER encoding of the RLRE APDU is shown in Table 138.

**Table 138 – BER encoding of the RLRE APDU**

-- BER encoding of the RLRE APDU	
// encoding of the tag of the RLRE APDU ([APPLICATION 3], Application)	63
// encoding of the length of the RLRE's contents field	28
-- reason field	
// encoding the tag ([0], IMPLICIT)	80
// encoding of the length of the tagged component's value field	01
// encoding of the value (0, normal)	00
-- encoding the user-information field component (Association-information, OCTET STRING)	
// encoding of the tag ([30], Context-specific, Constructed)	BE
// encoding of the length of the tagged component's value field	23

// encoding of the choice for user-information ( <b>OCTET STRING</b> , Universal)	04
// encoding of the length of the <b>OCTET STRING's</b> value field (14 octets)	21
// user-information: xDLMS InitiateResponse APDU	281F300123456789 1214A0845E475714 383F65BC19745CA2 35906525E4F3E1C8 93

## 13 S-FSK PLC encoding examples

### 13.1 CI-PDUs, ACSE APDUs and xDLMS APDUs carried by MAC frames using the IEC 61334-4-32 LLC sublayer

In these examples, the following communication sequence is shown, when the DLMS/COSEM S-FSK PLC profile is used with the IEC61334-4-32:1996 LLC sublayer:

- the initiator Discovers, then Registers a new server system;
- the initiator establishes an AA;
- it reads the time attribute of the Clock object (once and 13 times, to show block transfer);
- the initiator Pings a server;
- the initiator sends a RepeaterCall service.

In these examples: SYSTEM-TITLE-SIZE = 6.

The traces have been taken from a protocol analyser. The contents of the MAC frame are explained. The MAC frame is shown between the brackets () following the “02 xx 50” header and followed by 00 00 (final field, normally a frame check). The Pad fields are not shown.

Server in the NEW state with one alarm (Meter New) : MAC frame carrying a Discover CI-PDU

```
17:15:35:645 ==> Discover.Request(MAC:C00/FFF Ic:7 Dc:0 LLC:0/1) (Prob:100
NbTslot:10 CreditReponse:0 ICequalCredit:0)
Hex: 02 11 50 ( FC C0 0F FF 11 90 00 01 1D 64 00 0A 00 00 ) 00 00
```

-- Explanation:

```
FC // Credit fields: 1111 1100 IC = 7, CC = 7, DC = 0
C0 0F FF // MAC addresses: SA = C00 (Initiator), DA = FFF (All-
Physical)
11 // Pad length
90 // Control byte 1001 0000, DL-Data.request
00 01 // L-SAPs: DA = 00 (CIASE server), SA SAP = 01 (CIASE client)
1D // DiscoverRequest PDU
    64 // response-probability = 100
    00 0A // allowed-time-slots = 10
    00 // DiscoverReport-initial_credit = 00
    00 // ICEqualCredit = 00
```

MAC frame carrying a discoverReport CI-PDU

```
17:15:40:441 <== Alarm.Report(MAC:FFE/FFF Ic:0 Dc:0 LLC:FD/0) SN:
040890000001
Hex: 02 15 50 ( 00 FF EF FF 0D 90 FD 00 1E 01 04 08 90 00 00 01 01 01 ) 00 00
```

-- Explanation:

```
00 // Credit fields
FF EF FF // MAC addresses: SA = FFE (NEW), DA address = FFF
0D // Pad length
90 // DL-Data.request
FD 00 // L-SAPs: DA = FD, SA = 00
1E // discoverReport CI-PDU
    01 // SEQUENCE OF 1
        04 08 90 00 00 01 // System-Title
        01 // Alarm-Descriptor presence flag
        01 // Alarm-Descriptor
```

Register service: MAC frame carrying a Register CI-PDU

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	519/614
-----------------------	------------	-----------------------	---------

```
17:15:41:129 ==> Register(MAC:C00/FFF Ic:7 Dc:0 LLC:0/1) (AddrMAC: 0x3 SN:
040890000001 )
Hex: 02 1B 50 ( FC C0 OF FF 07 90 00 01 1C 04 08 99 00 00 01 01 04 08 90 00
00 01 00 03 ) 00 00
```

-- Explanation:

```
FC // Credit fields: 1111 1100 IC = 7, CC = 7, DC = 0
C0 OF FF // MAC addressees: SA = C00, DA = FFF
07 // Pad length
90 // DL-Data.request
00 01 // L-SAPs: DA = 00, SA = 01
1C // RegisterRequest CI-PDU
    04 08 99 00 00 01 // active-initiator-system-title
    01 // SEQUENCE OF 1
        04 08 90 00 00 01 // new-system-title
        00 03 // mac-address 0x03
```

Server in registered state with an alarm: MAC frame carrying a DiscoverRequest CI-PDU

```
17:17:02:973 ==> Discover.Request(MAC:C00/FFF Ic:7 Dc:0 LLC:0/1) (Prob:100
NbTslot:10 CreditReponse:0 ICequalCredit:0)
Hex: 02 11 50 ( FC C0 OF FF 11 90 00 01 1D 64 00 0A 00 00 ) 00 00
```

-- Explanation:

```
FC // Credit fields: 1111 1100 IC = 7, CC = 7, DC = 0
C0 OF FF // MAC addresses: SA = C00, DA = FFF
11 // Pad length
90 // DL-Data.request
00 01 // L-SAPs: DA = 00, SA = 01
1D // DiscoverRequest CI-PDU
    64 // response-probability = 100
    00 0A // allowed-time-slots 10
    00 // DiscoverReport-Initial-Credit 00
    00 // ICEqualCredit 0
```

Response: MAC frame carrying a DiscoverResponse CI-PDU

```
17:17:07:316 <== Alarm.Report(MAC:003/FFF Ic:0 Dc:0 LLC:FD/0) SN:
040890000001
Hex: 02 15 50 ( 00 00 3F FF 0D 90 FD 00 1E 01 04 08 90 00 00 01 01 82 ) 00 00
```

-- Explanation:

```
00 // Credit fields
00 3F FF // MAC addresses: SA = 003, DA = FFF
0D // Pad length
90 // DL-Data.request
FD 00 // L-SAPs: DA = FD, SA = 00
1E // discoverReport CI-PDU
    01 // SEQUENCE OF 1
        04 08 90 00 00 01 // System-Title
        01 // alarm-descriptor presence flag
        82 // alarm-descriptor
```

Open association on the Logical device LsapDest=0x01 and Client R/W LsapSrc=0x02: MAC frame carrying an AARQ APDU

```
17:28:52:691 ==> AARQ.Request(MAC:C00/003 Ic:4 Dc:0 LLC:1/2)
```

```
Hex: 02 43 50 ( 90 C0 00 03 03 90 01 02 60 36 A1 09 06 07 60 85 74 05 08 01
02 8A 02 07 80 8B 07 60 85 74 05 08 02 01 AC 0A 80 08 31 32 33 34 35 36 37 38
BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 1C 1A 20 00 EF ) 00 00
```

--Explanation:

```
90 // Credit fields
C0 00 03 // MAC addresses: SA = C00, DA = 003
03 // Pad length
90 // DL-Data.request
01 02 // L-SAPs: DA = 0x01, SA = 0x02
60 36 // AARQ APDU
    A1 09 06 07 60 85 74 05 08 01 02 // application-context-name
    8A 02 07 80 // acse-requirements
    8B 07 60 85 74 05 08 02 01 // mechanism-name
    AC 0A 80 08 31 32 33 34 35 36 37 38 // calling-authentication-
                                                value
    BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 1C 1A 20 00 EF
// user-information xDLMS InitiateRequest
```

Response: MAC frame carrying an AARE APDU

```
17:28:54:191 <== AARE.Response (MAC:003/C00 Ic:4 Dc:0 LLC:2/1)
Hex: 02 37 50 ( 90 00 3C 00 0F 90 02 01 61 29 A1 09 06 07 60 85 74 05 08 01
02 A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 10 04 0E 08 00 06 5F 1F 04 00 1C 1A
20 00 EF FA 00 00 ) 00 00
```

-- Explanation:

```
90 // Credit fields
00 3C 00 // MAC addresses: SA = 003, DA = C00
0F // Pad length
90 // DL-Data.request
02 01 L-SAPs: DA = 0x02, SA = 0x01
61 29 // AARE APDU
    A1 09 06 07 60 85 74 05 08 01 02 // application-context-name
    A2 03 02 01 00 // result
    A3 05 A1 03 02 01 00 // result-source-diagnostic
    BE 10 04 0E 08 00 06 5F 1F 04 00 1C 1A 20 00 EF FA 00 00
// user-information xDLMS-InitiateResponse
```

Read date and time current (1 short name)

```
17:35:16:082 ==> Read.Request[1] (7304) (MAC:C00/003 Ic:3 Dc:0 LLC:1/2)
Hex: 02 10 50 ( 6C C0 00 03 12 90 01 02 05 01 02 1C 88 ) 00 00
```

-- Explanation:

```
6C // Credit fields
C0 00 03 // MAC addresses
12 // Pad length
90 // DL-Data.request
01 02 // L-SAPs
05 01 // ReadRequest
    02 1C 88 // Variable-Name 1C88
```

```
17:35:16:832 <== Read.Response[1] (MAC:003/C00 Ic:3 Dc:0 LLC:2/1) ObjACMM:
0x1C88 (7304) | {2009/06/22 FF 17:35:15:FF 8000 FF}
Hex: 02 1C 50 ( 6C 00 3C 00 06 90 02 01 0C 01 00 09 0C 07 D9 06 16 FF 11 23
0F FF 80 00 FF ) 00 00
```

-- Explanation:

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	521/614
-----------------------	------------	-----------------------	---------

```

6C // Credit fields
00 3C 00 // MAC addresses
06 // Pad length
90 // DL-Data.request
02 01 // L-SAPs
0C 01 // ReadResponse
00 // Success
09 0C 07 D9 06 16 FF 11 23 0F FF 80 00 FF // value of the attribute

```

Read date and time current (13 short name, to provoke block transfer):

```

17:36:38:406 ==> Read.Request[13] (7304) (MAC:C00/003 Ic:0 Dc:0 LLC:1/2) |
ObjACMM: 0x1C88 (7304) | 0x1C88 (7304) | 0x1C88 (7304) | 0x1C88 (7304) |
0x1C88 (7304) | 0x1C88 (7304) | 0x1C88 (7304) | 0x1C88 (7304) | 0x1C88 (7304) |
| 0x1C88 (7304) | 0x1C88 (7304) | 0x1C88 (7304) | 0x1C88 (7304) |
Hex: 02 34 50 ( 00 C0 00 03 12 90 01 02 05 0D 02 1C 88 02 1C 88 02 1C 88 02
1C 88 02 1C 88 02 1C 88 02 1C 88 02 1C 88 02 1C 88 02 1C 88 02 1C 88 02
02 1C 88 ) 00 00

```

--Explanation:

```

00 // Credit fields
C0 00 03 // MAC addresses
12 // Pad length
90 // DL-Data.request
01 02 // L-SAPs
05 0D // Read 13 Variable-Access-Specification
    02 1C 88 // variable-name 1C 88
    02 1C 88 02 1C 88 02 1C 88 02 1C 88 02 1C 88 02 1C 88
    02 1C 88 02 1C 88 02 1C 88 02 1C 88 02 1C 88

```

```

17:36:39:609 <== Read.Response DataBlock_DLMS (MAC:003/C00 Ic:0 Dc:0 LLC:2/1)
LastBlock:0 Block:1
Hex: 02 91 50 ( 00 00 3C 00 21 90 02 01 0C 01 02 00 00 01 81 7E 0D 00 09 0C
07 D9 06 16 FF 11 24 25 FF 80 00 FF 00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00
FF 00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF 00 09 0C 07 D9 06 16 FF 11 24
25 FF 80 00 FF 00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF 00 09 0C 07 D9 06
16 FF 11 24 25 FF 80 00 FF 00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF 00 09
0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF 00 09 0C 07 D9 ) 00 00

```

--Explanation:

```

00 // Credit fields
00 3C 00 // MAC addresses
21 // Pad length
90 // DL-Data.request
02 01 // L-SAPs
0C 01 02 // ReadResponse data-block-result
    00 // last-block = FALSE
    00 01 // block-number 00 01
    81 7E // raw-data octet-string of 126 bytes
    0D // 13 results
    00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
    // first result success, attribute value
    00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
    // second result success, attribute value
    00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
    00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
    00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
    00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF

```

```

00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
00 09 0C 07 D9 // end of the first block

17:36:40:047 ==> ReadNextBlock.Request[1] (MAC:C00/003 Ic:0 Dc:0 LLC:1/2)
Block:1
Hex: 02 10 50 ( 00 C0 00 03 12 90 01 02 05 01 05 00 01 ) 00 00

00 // Credit fields
C0 00 03 // MAC addresses
12 // pad length
90 // DL-Data.request
01 02 // L-SAPs
05 01 05 // ReadRequest, variable-access-specification, block-number-
           // access
00 01 // block-number 00 01

17:36:40:797 <== Read.Response DataBlock_DLMS (MAC:003/C00 Ic:0 Dc:0 LLC:2/1)
LastBlock:1 Block:2
Hex: 02 58 50 ( 00 00 3C 00 12 90 02 01 0C 01 02 01 00 02 46 06 16 FF 11 24
25 FF 80 00 FF 00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF 00 09 0C 07 D9 06
16 FF 11 24 25 FF 80 00 FF 00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF 00 09
0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF ) 00 00

-- Explanation:

00 // Credit fields
00 3C 00 // MAC addresses
12 // Pad length
90 // DL-Data.request
02 01 // L-SAPs
0C 01 02 // ReadResponse data-block-result
          01 // last-block = TRUE
          00 02 // block-number = 00 02
          46 // octet-string of 70 bytes
                  06 16 FF 11 24 25 FF 80 00 FF
00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF
00 09 0C 07 D9 06 16 FF 11 24 25 FF 80 00 FF

```

#### Ping service: MAC frame carrying a pingRequest CI-PDU

```

17:38:43:633 ==> Ping.Request(MAC:C00/003 Ic:0 Dc:0 LLC:0/1 |SN: 04 08 90 00
00 01)
Hex: 02 12 50 ( 00 C0 00 03 10 90 00 01 19 04 08 90 00 00 01 ) 00 00

```

-- Explanation:

```

00 // Credit fields
C0 00 03 // MAC addresses
10 // Pad length
90 //DL-Data.request
00 01 // L-SAPs
19 // pingRequest CI-PDU
      04 08 90 00 00 01 // System-Title

```

#### Response: MAC frame carrying a PingResponse CI-PDU

```
17:38:44:383 <== Ping.Response(MAC:003/C00 Ic:0 Dc:0 LLC:1/0 |SN: 04 08 90
00 00 01)
Hex: 02 12 50 ( 00 00 3C 00 10 90 01 00 1A 04 08 90 00 00 01 ) 00 00
```

-- Explanation:

```
00 // Credit fields
00 3C 00 // MAC addresses
10 // Pad length
90 // DL-Data.request
01 00 // L-SAPs
1A // pingResponse CI-PDU
    04 08 90 00 00 01 // System-Title
```

#### RepeaterCall service

```
17:38:54:727 ==> RepeaterCall(MAC:C00/FFF Ic:7 Dc:0 LLC:0/1) Max_Adress_MAC:
0x63 Nb_Tslot_For_NEW: 0
Hex: 02 10 50 ( FC C0 0F FF 12 90 00 01 1F 00 63 00 00 ) 00 00
```

-- Explanation:

```
FC // Credit fields
C0 0F FF // MAC addresses: SA = C00, DA = FFF
12 // Pad length
90 // DL-Data.request
00 01 // L-SAPs: DA = 00, SA = 01
1F // RepeaterCall CI-PDU
    00 63 // MaxAdrMac 0x63
    00 // Nb_Tslot_For_New = 0
    00 // Reception-Threshold default value
```

### 13.2 CI-PDUs, ACSE APDUs and xDLMS APDUs carried by MAC frames using the HDLC based LLC sublayer

In these examples, the following communication sequence is shown, when the DLMS/COSEM S-FSK PLC profile is used with the HDLC based LLC sublayer:

- the initiator Discovers, then Registers a new server system;
- it connects the HDLC based LLC sublayer and establishes an AA;
- it reads the time attribute of the Clock object;
- it releases the AA by disconnecting the HDLC based LLC sublayer.

In these examples: SYSTEM-TITLE-SIZE = 8.

```
-- The following trace is a spy frame of a chip implementing IEC 61334-5-1.
2009-05-14 16:04:53.922686 IEC61334-5-1-SPY [SPY-SUBFRAME] LEN=55
S0/N0=7928/164 S1/N1=4654/374 THR=27 MET=4 SYN=0 RGAIN=2 P_SDU_LEN=38
-- Spy frame carrying a Phy frame. The Spy frame is not part of this Technical Report.
```

```
0000 02 35 B0 F8 1E A4 00 2E 12 76 01 1B 00 04 02 00
0010 00 6C 6C 00 C0 1F FF 05 7E A0 13 CE FF CD 13 61
0020 D5 E6 E6 00 1D 64 00 14 00 00 2C 66 7E 00 00 00
0030 00 00 32 9B EA 6E 10
```

-- Explanation:

```
02 // STX
35 // length
B0 // Spy subframe
F8 1E A4 00 2E 12 76 01 1B 00 04 02 // Spy data
-- here follows the 38 bytes Phy frame, carrying the MAC frame
00 00 6C 6C 00 C0 1F FF 05 7E A0 13 CE FF CD 13
```

```

61 D5 E6 E6 00 1D 64 00 14 00 00 2C 66 7E 00 00
00 00 00 32 9B EA
-- end of Phy frame
6E 10 // spy frame check field

```

Discover service: MAC frame carrying a Discover CI-PDU

```

-- For the MAC frame format, see IEC 61334-4-32:1996, 4.2.2
0000 6C 6C 00 C0 1F FF 05 7E A0 13 CE FF CD 13 61 D5
0010 E6 E6 00 1D 64 00 14 00 00 2C 66 7E 00 00 00 00
0020 00 32 9B EA

```

-- Explanation:

```

6C 6C // NS field, number of MAC subframes is 1
00 // Credit fields, IC = 0, CC = 0, DC = 0
C0 1F FF // MAC addresses: SA = C01, Initiator, DA = FFF
// All-Physical
05 // Pad length
7E // HDLC frame flag
A0 13 // Frame type and length
CE FF CD // MAC addresses: DA = 0x677F, upper HDLC address 0x67, lower HDLC
address = All-station, SA = 0x66
13 // UI frame
61 D5 // HDLC HCS
E6 E6 00 // DLMS/COSEM LLC addresses
1D // Discover CI-PDU
    64 // response-probability = 100
    00 14 // allowed-time-slots = 20
    00 // DiscoverReport-initial-credit = 0
    00 // ICEqualCredit = 0
2C 66 // HDLC FCS
7E // HDLC frame flag
00 00 00 00 00 // padding
32 9B EA // MAC FCS

```

-- From here on, only the MAC frames are shown and explained

Response: MAC frame carrying a discoverReport CI-PDU

```

0000 6C 6C 00 FF EC 01 00 7E A0 18 CD CE 23 13 BB 18
0010 E6 E7 00 1E 01 49 53 4B 05 00 00 00 01 00 B3 01
0020 7E 38 CD 0F

```

-- Explanation:

```

6C 6C // NS field, number of MAC subframes is 1
00 // Credit fields, IC = 0, CC = 0, DC = 0
FF EC 01 // MAC addresses: SA = FFE (NEW), DA = C01, Initiator
00 // Pad length
7E // HDLC frame flag

```

**A0 18 // Frame type and length**

```

CD CE 23 // DA = 0x66, SA = 0x6711, 0x11 is the lower HDLC address of
          the system sending the discoverReport
13 // UI frame
BB 18 // HDLC HCS
E6 E7 00 // DLMS/COSEM LLC addresses
-- discoverReport CI-PDU
    1E // discoverReport CI-PDU tag [30]
    01 // Sequence of 1
        49 53 4B 05 00 00 00 01 // system-title-server
        00 // Presence flag of the alarm-descriptor, not present
B3 01 // HDLC FCS
7E // HDLC frame flag

```

38 CD 0F // MAC FCS

Register service: MAC frame carrying a Register CI-PDU

```

0000  3A 3A 00 C0 1F FF 1B 7E A0 21 CE FF CD 13 38 17
0010  E6 E6 00 1C FE FE FE FE FE FE FE 01 49 53 4B
0020  05 00 00 00 01 00 10 0C E6 7E 00 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040  00 00 00 00 54 F2 23

```

-- Explanation:

```

3A 3A // NS field, number of MAC subframes is 2
00 // Credit fields, IC = 0, CC = 0, DC = 0
C0 1F FF // MAC addresses: SA = C01, Initiator, DA = FFF, All-Physical
1B // Pad length, 27 bytes
7E // HDLC frame flag
A0 21 // Frame type and length
CE FF CD // DA = 0x677F, upper HDLC address All-station, SA = 66
13 // UI frame
38 17 // HDLC HCS
E6 E6 00 // DLMS/COSEM LLC addresses
1C // Register CI-PDU tag
    FE FE FE FE FE FE FE // active-initiator-system-title
    01 // sequence of 1
        49 53 4B 05 00 00 00 01 // system-title-server
    00 10 // MAC-address
0C E6 // HDLC FCS
7E // HDLC frame flag
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
54 F2 23 // MAC FCS

```

Establishment of a data link layer connection: MAC frame carrying an SNRM HDLC frame

```

0000  6C 6C 00 C0 10 10 10 7E A0 08 02 23 C9 93 E4 43
0010  7E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020  00 3F 96 F1

```

-- Explanation:

```

6C 6C // NS field, number of MAC subframes is 1
00 // Credit fields, IC = 0, CC = 0, DC = 0
C0 10 10 // MAC addresses: SA = C01, Initiator, DA = 010, Individual
10 // Pad length
7E // HDLC frame flag
A0 08 // Frame type and length
02 23 C9 // DA = 0x0111, SA = 0x64
93 // SNRM frame
E4 43 // HDLC FCS
7E // // HDLC frame flag
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
3F 96 F1 // MAC FCS

```

Response: MAC frame carrying a HDLC UA frame

```

0000  3A 3A 00 01 0C 01 1D 7E A0 1F C9 02 23 73 B4 96
0010  81 80 12 05 01 7E 06 01 7E 07 04 00 00 00 01 08
0020  04 00 00 00 01 5F 75 7E 00 00 00 00 00 00 00 00 00
0030  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040  00 00 00 00 72 3D 01

```

```
-- Explanation:
3A 3A // NS field, number of MAC subframes is 2
00 // Credit fields, IC = 0, CC = 0, DC = 0
01 0C 01 // MAC addresses: SA = 010, Individual, DA = C01, Initiator
1D // pad length
7E // HDLC frame flag
A0 1F // Frame type and length
C9 02 23 // SA = 0x64, DA = 0x0111
73 // UA frame
B4 96 // HDLC HCS
81 80 12 05 01 7E 06 01 7E 07 04 00 00 00 00 01 08
04 00 00 00 01 // information field
5F 75 // HDLC FCS
7E // HDLC frame flag
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // padding
72 3D 01 //MAC FCS
```

Establishment of an AA: MAC frame carrying an AARQ APDU

0000	56 56 00 C0 10 10 1B 7E A0 45 02 23 C9 10 21 48
0010	E6 E6 00 60 36 A1 09 06 07 60 85 74 05 08 01 01
0020	8A 02 07 80 8B 07 60 85 74 05 08 02 01 AC 0A 80
0030	08 31 32 33 34 35 36 37 38 BE 10 04 0E 01 00 00
0040	00 06 5F 1F 04 00 00 7E 1F FF FF 83 D7 7E 00 00
0050	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0060	00 00 00 00 00 00 00 00 00 00 00 00 9B FF 67

```
-- Explanation:
56 56 // NS field, number of MAC subframes is 3
00 // Credit fields, IC = 0, CC = 0, DC = 0
C0 10 10 // MAC addresses SA = C01, Initiator, DA = 010, Individual
1B // Pad length
7E // HDLC frame flag
A0 45 // Frame type and length
02 23 C9 // DA = 0x0111, SA = 0x64
10 // I frame
21 48 // HDLC HCS
E6 E6 00 // LLC addresses
60 36 // AARQ APDU
    A1 09 06 07 60 85 74 05 08 01 01 // application-context-name
    8A 02 07 80 // acse-requirements
    8B 07 60 85 74 05 08 02 01 // mechanism-name
    AC 0A 80 08 31 32 33 34 35 36 37 38 // calling-authentication-
                                            value
    BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F FF FF
    // user-information xDLMS-Initiate.request
83 D7 // HDLC FCS
7E // HDLC frame flag
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
9B FF 67
```

Response: MAC frame carrying an AARE APDU

0000	3A 3A 00 01 0C 01 03 7E A0 39 C9 02 23 30 22 BD
0010	E6 E7 00 61 2A A1 09 06 07 60 85 74 05 08 01 01
0020	A2 03 02 01 00 A3 05 A1 03 02 01 00 BE 11 04 0F
0030	08 01 00 06 5F 1F 04 00 00 7C 1F 04 00 00 07 19
0040	4A 7E 00 00 10 E9 9A

```
-- Explanation:
3A 3A // NS field, number of MAC subframes is 2
00 // Credit fields, IC = 0, CC = 0, DC = 0
01 0C 01 // MAC addresses: SA = 010 Individual, DA = 010, Initiator
03 // pad length
7E // HDLC frame flag
A0 39 // Frame type and length
C9 02 23 // SA = 0x64, DA = 0x0111
30 // HDLC I frame
22 BD // HDLC HCS
E6 E7 00 // LLC addresses
61 2A // AARE APDU
    A1 09 06 07 60 85 74 05 08 01 01 // application-context-name
    A2 03 02 01 00 // result
    A3 05 A1 03 02 01 00 // result-source-diagnostic
    BE 11 04 0F 08 01 00 06 5F 1F 04 00 00 7C 1F 04 00 00 07
    // user-information xDLMS-Initiate.response
19 4A // HDLC FCS
7E // HDLC frame flag
00 00 00 // pad
10 E9 9A // MAC FCS
```

Get-request-normal APDU

0000	3A 3A 00 C0 10 10 22 7E A0 1A 02 23 C9 32 AF 55
0010	E6 E6 00 C0 01 40 00 08 00 00 01 00 00 FF 02 00
0020	EA DD 7E 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040	00 00 00 00 00 C2 2B 4A

```
-- Explanation:
3A 3A // NS field, number of MAC subframes is 2
00 // Credit fields, IC = 0, CC = 0, DC = 0
C0 10 10 // MAC addresses: SA = C01, Initiator, DA = 010, Individual
22 // pad length
7E // HDLC frame flag
A0 1A // Frame type and length
02 23 C9 // DA = 0x0111, SA = 0x64
32 // HDLC I frame
AF 55 // HDLC HCS
E6 E6 00 // LLC addresses
C0 01 40 00 08 00 00 01 00 00 FF 02 00 // Get-request-normal APDU
EA DD // HDLC FCS
7E // HDLC frame flag
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 // PAD
C2 2B 4A // MAC FCS
```

Get-response-normal APDU

0000	3A 3A 00 01 0C 01 1D 7E A0 1F C9 02 23 52 3F A6
0010	E6 E7 00 C4 01 40 00 09 0C 07 D2 01 07 01 01 23
0020	1A 00 FF C4 00 80 EC 7E 00 00 00 00 00 00 00 00 00
0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040	00 00 00 00 00 C1 62 A6

```
-- Explanation:
3A 3A // NS field, number of MAC subframes is 2
00 // Credit fields, IC = 0, CC = 0, DC = 0
01 0C 01 // MAC addresses: SA = 010 Individual, DA = 010, Initiator
```

```

1D / Pad length
7E // HDLC frame flag
A0 1F // Frame type and length
C9 02 23 // SA = 0x64, DA = 0x0111
52 // HDLC I frame
3F A6 // HDLC HCS
E6 E7 00 // LLC addresses
-- Get-response-normal APDU
C4 01 40 00 09 0C 07 D2 01 07 01 01 23 1A 00 FF C4 00
80 EC // HDLC FCS
7E // HDLC frame flag
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
C1 62 A6 // MAC FCS

```

Releasing the AA: MAC frame carrying a HDLC DISC frame

```

0000 6C 6C 00 C0 10 10 10 7E A0 08 02 23 C9 53 E8 85
0010 7E 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0020 00 B9 A4 CD

```

```

6C 6C // NS field, number of MAC subframes is 1
00 // Credit fields, IC = 0, CC = 0, DC = 0
C0 10 10 // MAC addresses: SA = C01, Initiator, DA = 010, Individual
10 // pad length
7E // HDLC frame flag
A0 08 // Frame type and length
02 23 C9 // DA = 0x0111, SA = 0x64
53 // HDLC DISC frame
E8 85 // HDLC FCS
7E // HDLC frame flag
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // pad
B9 A4 CD // MAC FCS

```

Response: MAC frame carrying a HDLC UA frame

```

0000 3A 3A 00 01 0C 01 1D 7E A0 1F C9 02 23 73 B4 96
0010 81 80 12 05 01 7E 06 01 7E 07 04 00 00 00 00 01 08
0020 04 00 00 00 01 5F 75 7E 00 00 00 00 00 00 00 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0040 00 00 00 00 00 72 3D 01
3A 3A // NS field, number of MAC subframes is 2
00
01 0C 01 // MAC addresses: SA = 010 Individual, DA = 010, Initiator
1D // Pad length
7E // HDLC frame flag
A0 1F // Frame type and length
C9 02 23 // SA = 0x64, DA = 0x0111
73 // HDLC UA frame
B4 96 // HDLC HCS
// Information field
81 80 12 05 01 7E 06 01 7E 07 04 00 00 00 01 08 04 00 00 00 01
5F 75 // HDLC FCS
7E // HDLC frame flag
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 // pad
72 3D 01 // MAC FCS

```

**13.3 Clear Alarm examples**

In these examples, SYSTEM-TITLE-SIZE = 6.

Example 1: Clearing a single alarm in all servers

```
39 // tag for clearAlarm, [57]
00 // Choice 0, Alarm-descriptor
00 // Alarm-Descriptor, fixed length unsigned integer
```

Example 2: Clearing a list of alarms in all servers

```
39 // tag for ClearAlarm, [57]
01 // CHOICE 1, alarm-descriptor-list, SEQUENCE OF Alarm-Descriptor
01 // Number of elements in the SEQUENCE OF
00 // Contents field: Alarm-Descriptor, fixed length unsigned integer
```

Example 3: Clearing a list of alarms in some servers

```
39 / tag for clearAlarm, [57]
02 // CHOICE 2, SEQUENCE Alarm-Descriptor-List-And-Server-List
01 // server-id-list, number of elements in the SEQUENCE OF System-Title
040967000001 // System-title, fixed length octet-string
01 // alarm-descriptor-list, number of elements in the SEQUENCE OF Alar,-
Descriptor
00 // Alarm-Descriptor, fixed length unsigned integer
```

Example 4: Clearing a different alarm in each different server

```
39 // tag for clearAlarm, [57]
03 // CHOICE 3, alarm-descriptor-by-server-list
01 // SEQUENCE OF Alarm-Descriptor-By-Server
040967000001 // First element of the SEQUENCE: System-Title
00 // Second element of the SEQUENCE: Alarm-Descriptor
```

## 14 Data transfer service examples

### 14.1 GET / Read, SET / Write examples

Table 140 to Table 147 show examples for data exchange using xDLMS services with LN referencing (left column) and SN referencing (right column). Table 139 shows the objects used in the examples.

**Table 139 – The objects used in the examples**

Object 1: <ul style="list-style-type: none"> <li>- Class: Data</li> <li>- Logical name: 0000800000FF</li> <li>- Short name of value attribute: 0100</li> <li>- Value: octet string of 50 elements</li> <li>- 01020304050607080910111213141516</li> <li>- 17181920212223242526272829303132</li> <li>- 33343536373839404142434445464748</li> <li>- 4950</li> </ul> Object 2: <ul style="list-style-type: none"> <li>- Class: Data</li> <li>- Logical name: 0000800100FF</li> <li>- Short name of value attribute: 0110</li> <li>- Value: visible string of 3 elements 303030</li> </ul>
--

In the case of block transfer, the negotiated APDU size is 40 bytes.

Nota bene: What is negotiated is the APDU size not the block size! Therefore, the block size is smaller than the APDU size.

**Table 140 – Example: Reading the value of a single attribute without block transfer**

C001C1 0001000800000FF0200  <GetRequest> <GetRequestNormal> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </GetRequestNormal> </GetRequest>	0501 020100  <ReadRequest Qty="0001" > <VariableName Value="0100" /> </ReadRequest>
C401C1 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950  <GetResponse> <GetResponsenormal> <InvokeIdAndPriority Value="C1" /> <Result> <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> </Result> </GetResponsenormal> </GetResponse>	0C01 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950  <ReadResponse Qty="0001" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> </ReadResponse>

**Table 141 – Example: Reading the value of a list of attributes without block transfer**

C003C1 02 00010000800000FF0200 00010000800100FF0200  <GetRequestWithList> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptorList Qty="0002" > <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </_AttributeDescriptorWithSelection> <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800100FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </_AttributeDescriptorWithSelection> </AttributeDescriptorList> </GetRequestWithList> </GetRequest>	0502 020100 020110  <ReadRequest Qty="0002" > <VariableName Value="0100" /> <VariableName Value="0110" /> </ReadRequest>
---	---

C403C1 02 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 00 0A03 303030  <GetResponse> <GetResponseWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> <Data> <VisibleString Value="303030" /> </Data> </Result> </GetResponseWithList> <br&gt;&lt; getresponse&gt;<="" td=""><td>0C02 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 00 0A03 303030  &lt;ReadResponse Qty="0002" &gt;   &lt;Data&gt;     &lt;OctetString Value="01020304050607080910111213141516       17181920212223242526272829303132       33343536373839404142434445464748       4950" /&gt;   &lt;/Data&gt;   &lt;Data&gt;     &lt;VisibleString Value="303030" /&gt;   &lt;/Data&gt;<br&gt;&lt; readresponse&gt;<="" td=""></br&gt;&lt;></td></br&gt;&lt;>	0C02 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 00 0A03 303030  <ReadResponse Qty="0002" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> <Data> <VisibleString Value="303030" /> </Data> <br&gt;&lt; readresponse&gt;<="" td=""></br&gt;&lt;>
--	---

**Table 142 – Example: Reading the value of a single attribute with block transfer**

C001C1 00010000800000FF0200  <GetRequest> <GetRequestNormal> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </GetRequestNormal> <br&gt;&lt; getrequest&gt;<="" td=""><td>0501 020100  &lt;ReadRequest Qty="0001" &gt;   &lt;VariableName Value="0100" /&gt;<br &gt;&lt;readrequest&gt;<="" td=""/></td></br&gt;&lt;>	0501 020100  <ReadRequest Qty="0001" > <VariableName Value="0100" /> 
--	--

<pre>C402C1 00 00000001 00 1E 093201020304050607080910111213 141516171819202122232425262728  &lt;GetResponse&gt;   &lt;GetResponsewithDataBlock&gt;     &lt;InvokeIdAndPriority Value="C1" /&gt;     &lt;Result&gt;       &lt;LastBlock Value="00" /&gt;       &lt;BlockNumber Value="00000001" /&gt;       &lt;Result&gt;         &lt;RawData Value="09320102030405060708091011121314                       1516171819202122232425262728" /&gt;       &lt;/Result&gt;     &lt;/Result&gt;   &lt;/GetResponsewithDataBlock&gt; &lt;/GetResponse&gt; // 30 bytes of raw-data contains data type, length and 28 bytes // of data. Note that Data is encoded, not Get-Data-result.</pre>	<pre>0C01 02 00 0001 21 01000932010203040506070809101112 13141516171819202122232425262728 29  &lt;ReadResponse Qty="0001" &gt;   &lt;DataBlockResult&gt;     &lt;LastBlock Value="00" /&gt;     &lt;BlockNumber Value="0001" /&gt;     &lt;RawData Value="01000932010203040506070809101112                   13141516171819202122232425262728                   29" /&gt;   &lt;/DataBlockResult&gt; &lt;/ReadResponse&gt;  // 33 bytes of raw-data contains number of data, success, data // type, length and 29 bytes of data. // As the raw-data contains the data encoded exactly as without // block transfer, the number of results is encoded because the // ReadResponse is a <b>SEQUENCE OF CHOICE</b>.</pre>
<pre>C002C1 00000001  &lt;GetRequest&gt;   &lt;GetRequestForNextDataBlock&gt;     &lt;InvokeIdAndPriority Value="C1" /&gt;     &lt;BlockNumber Value="00000001" /&gt;   &lt;/GetRequestForNextDataBlock&gt; &lt;/GetRequest&gt;</pre>	<pre>0501 050001  &lt;ReadRequest Qty="0001" &gt;   &lt;BlockNumberAccess&gt;     &lt;BlockNumber Value="0001" /&gt;   &lt;/BlockNumberAccess&gt; &lt;/ReadRequest&gt;</pre>

<pre>C402C1 01 00000002 00 16 29303132333435363738394041424344 454647484950  &lt;GetResponse&gt;   &lt;GetResponsewithDataBlock&gt;     &lt;InvokeIdAndPriority Value="C1" /&gt;     &lt;Result&gt;       &lt;LastBlock Value="01" /&gt;       &lt;BlockNumber Value="00000002" /&gt;       &lt;Result&gt;         &lt;RawData Value="29303132333435363738394041424344           454647484950" /&gt;       &lt;/Result&gt;     &lt;/Result&gt;   &lt;/GetResponsewithDataBlock&gt; &lt;/GetResponse&gt;  // APDU length 32 bytes, 22 bytes of raw-data carries the // remaining part of data requested.</pre>	<pre>0C01 02 01 0002 15 30313233343536373839404142434445 4647484950  &lt;ReadResponse Qty="0001" &gt;   &lt;DataBlockResult&gt;     &lt;LastBlock Value="01" /&gt;     &lt;BlockNumber Value="0002" /&gt;     &lt;RawData Value="30313233343536373839404142434445       4647484950" /&gt;   &lt;/DataBlockResult&gt; &lt;/ReadResponse&gt;  // APDU length 28 bytes, 21 bytes of raw-data carries the // remaining part of data requested.</pre>
---	--

**Table 143 – Example: Reading the value of a list of attributes with block transfer**

C003C1 02 00010000800000FF0200 00010000800100FF0200  <GetRequestWithList> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptorList Qty="0002" > <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </_AttributeDescriptorWithSelection> <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800100FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </_AttributeDescriptorWithSelection> </AttributeDescriptorList> </GetRequestWithList> </GetRequest>	0502 020100 020110  <ReadRequest Qty="0002" > <VariableName Value="0100" /> <VariableName Value="0110" /> </ReadRequest>
---	---

C402C1 00 00000001 00 1E 02000932010203040506070809101112 1314151617181920212223242526  <GetResponse> <GetResponsewithDataBlock> <InvokeIdAndPriority Value="C1" /> <Result> <LastBlock Value="00" /> <BlockNumber Value="00000001" /> <Result> <RawData Value="02000932010203040506070809101112 13141516171819202122232425262728 29" /> </Result> </GetResponsewithDataBlock> </GetResponse>  // 30 bytes of raw-data contains the number of results and part // of the data. The first one is success, octet-string of 32 // elements; the first 26 bytes fit in.	0C01 02 00 0001 21 02000932010203040506070809101112 13141516171819202122232425262728 29  <ReadResponse Qty="0001" > <DataBlockResult> <LastBlock Value="00" /> <BlockNumber Value="0001" /> <RawData Value="02000932010203040506070809101112 13141516171819202122232425262728 29" /> </DataBlockResult> </ReadResponse>  // 33 bytes of raw-data contains the number of results and part // of the data. The first one is success, octet-string of 32 // elements; the first 29 bytes fit in.
C002C1 00000001  <GetRequest> <GetRequestForNextDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </GetRequestForNextDataBlock> </GetRequest>	0501 05 0001  <ReadRequest Qty="0001" > <BlockNumberAccess> <BlockNumber Value="0001" /> </BlockNumberAccess> </ReadRequest>

```
C402C1
01
00000002
00
1E
27282930313233343536373839404142
4344454647484950000A03303030

<GetResponse>
  <GetResponsewithDataBlock>
    <InvokeIdAndPriority Value="C1" />
    <Result>
      <LastBlock Value="01" />
      <BlockNumber Value="00000002" />
      <Result>
        <RawData Value="27282930313233343536373839404142
          4344454647484950000A03303030" />
      </Result>
    </Result>
  </GetResponsewithDataBlock>
</GetResponse>

// 30 bytes of raw-data contains the remaining 24 bytes of the
// first result and it also contains the six bytes of the second
// result: success, visible string of 3 elements.
```

```
0C01
02
01
0002
1B
30313233343536373839404142434445
4647484950000A03303030

<ReadResponse Qty="0001" >
  <DataBlockResult>
    <LastBlock Value="01" />
    <BlockNumber Value="0002" />
    <RawData Value="30313233343536373839404142434445
      4647484950000A03303030" />
  </DataBlockResult>
</ReadResponse>

// The APDU is 34 bytes. It contains the second and last block.
// 27 bytes of raw-data contains the remaining 21 bytes of the
// first result and the six bytes of the second result: success, //
visible string of 3 elements
```

**Table 144 – Example: Writing the value of a single attribute without block transfer**

C101C1 0001000800000FF0200 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950  <SetRequest> <SetRequestNormal> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> <Value> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Value> </SetRequestNormal> </SetRequest>	06 01 020100 01 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950  <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <VariableName Value="0100" /> </ListOfVariableAccessSpecification> <ListOfData Qty="0001" > <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </ListOfData> </WriteRequest>
C501C1 00  <SetResponse> <SetResponseNormal> <InvokeIdAndPriority Value="C1" /> <Result Value="Success" /> </SetResponseNormal> </SetResponse>	0D01 00  <WriteResponse Qty="0001" > <Success /> </WriteResponse>

**Table 145 – Example: Writing the value of a list of attributes without block transfer**

C104C1 02 00010000800000FF0200 00010000800100FF0200 02 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 0A03 303030  <SetRequest> <SetRequestNormalWithList> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptorList Qty="0002" > <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </_AttributeDescriptorWithSelection> <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800100FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </_AttributeDescriptorWithSelection> </AttributeDescriptorList> <ValueList Qty="0002" > <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> <VisibleString Value="303030" /> </ValueList> </SetRequestNormalWithList> </SetRequest>	0602 020100 020110 02 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 0A03 303030  <WriteRequest> <ListOfVariableAccessSpecification Qty="0002" > <VariableName Value="0100" /> <VariableName Value="0110" /> </ListOfVariableAccessSpecification> <ListOfData Qty="0002" > <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> <VisibleString Value="303030" /> </ListOfData> </WriteRequest>
--	---

C505C1 02 00 00  <SetResponse> <SetResponseWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <_DataAccessResult Value="Success" /> <_DataAccessResult Value="Success" /> </Result> </SetResponseWithList> <br&gt;&lt; setresponse&gt;<="" td=""><td>0D02 00 00  &lt;WriteResponse Qty="0002" &gt;   &lt;Success /&gt;   &lt;Success /&gt; &lt;/WriteResponse&gt;</td></br&gt;&lt;>	0D02 00 00  <WriteResponse Qty="0002" > <Success /> <Success /> </WriteResponse>
--	---

**Table 146 – Example: Writing the value of a single attribute with block transfer**

<pre>C102C1 00010000800000FF0200 00 00000001 15 09320102030405060708091011121314 1516171819  &lt;SetRequest&gt;   &lt;SetRequestWithFirstDataBlock&gt;     &lt;InvokeIdAndPriority Value="C1" /&gt;     &lt;AttributeDescriptor&gt;       &lt;ClassId Value="0001" /&gt;       &lt;InstanceId Value="0000800000FF" /&gt;       &lt;AttributeId Value="02" /&gt;     &lt;/AttributeDescriptor&gt;     &lt;DataBlock&gt;       &lt;LastBlock Value="00" /&gt;       &lt;BlockNumber Value="00000001" /&gt;       &lt;RawData Value="09320102030405060708091011121314                     1516171819" /&gt;     &lt;/DataBlock&gt;   &lt;/SetRequestWithFirstDataBlock&gt; &lt;/SetRequest&gt;  // 21 bytes of raw-data contain the type, length and the first 19 // bytes of data to be written.</pre>	<pre>0601 07 00 0001 01 091F 01020100010932010203040506070809 101112131415161718192021222324  &lt;WriteRequest&gt;   &lt;ListOfVariableAccessSpecification Qty="0001" &gt;     &lt;WriteDataBlockAccess&gt;       &lt;LastBlock Value="00" /&gt;       &lt;BlockNumber Value="0001" /&gt;     &lt;/WriteDataBlockAccess&gt;   &lt;/ListOfVariableAccessSpecification&gt;   &lt;ListOfData Qty="0001" &gt;     &lt;OctetString Value="01020100010932010203040506070809                     101112131415161718192021222324" /&gt;   &lt;/ListOfData&gt; &lt;/WriteRequest&gt;  // 31 bytes of octet-string contains raw-data: the sequence of // Variable-Access-Specification, the sequence of data, the type, // length and the first 24 bytes to be written.</pre>
--	---

C502C1 00000001  <SetResponse> <SetResponseForDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </SetResponseForDataBlock> </SetResponse>	0D01 02 0001  <WriteResponse Qty="0001" > <BlockNumber Value="0001" /> </WriteResponse>
C103C1 01 00000002 1F 20212223242526272829303132333435 363738394041424344454647484950  <SetRequest> <SetRequestWithDataBlock> <InvokeIdAndPriority Value="C1" /> <DataBlock> <LastBlock Value="01" /> <BlockNumber Value="00000002" /> <RawData Value="20212223242526272829303132333435 363738394041424344454647484950" /> </DataBlock> </SetRequestWithDataBlock> </SetRequest>  // 31 bytes of raw-data contains the remaining 21 bytes of the // data to be written.	0601 07 01 0002 01 091A 25262728293031323334353637383940 41424344454647484950  <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="01" /> <BlockNumber Value="0002" /> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> <ListOfData Qty="0001" > <OctetString Value="25262728293031323334353637383940 41424344454647484950" /> </ListOfData> </WriteRequest>  // The APDU is 35 bytes. 26 bytes of octet-string contains raw- // data: the remaining 26 bytes of data to be written.

## DLMS/COSEM Architecture and Protocols

---

C503C100000000002

```
<SetResponse>
  <SetResponseForLastDataBlock>
    <InvokeIdAndPriority Value="C1" />
    <Result Value="Success" />
    <BlockNumber Value="00000002" />
  </SetResponseForLastDataBlock>
</SetResponse>
```

0D0100

```
<WriteResponse Qty="0001" >
  <Success />
</WriteResponse>
```

**Table 147 – Example: Writing the value of a list of attributes with block transfer**

<pre> C105C1 02 00010000800000FF0200 00010000800100FF0200 00 00000001 0A 02093201020304050607  &lt;SetRequest&gt;   &lt;SetRequestWithListAndFirstDatablock&gt;     &lt;InvokeIdAndPriority Value="C1" /&gt;     &lt;AttributeDescriptorList Qty="0002" &gt;       &lt;_AttributeDescriptorWithSelection&gt;         &lt;AttributeDescriptor&gt;           &lt;ClassId Value="0001" /&gt;           &lt;InstanceId Value="0000800000FF" /&gt;           &lt;AttributeId Value="02" /&gt;         &lt;/AttributeDescriptor&gt;       &lt;/_AttributeDescriptorWithSelection&gt;       &lt;_AttributeDescriptorWithSelection&gt;         &lt;AttributeDescriptor&gt;           &lt;ClassId Value="0001" /&gt;           &lt;InstanceId Value="0000800100FF" /&gt;           &lt;AttributeId Value="02" /&gt;         &lt;/AttributeDescriptor&gt;       &lt;/_AttributeDescriptorWithSelection&gt;     &lt;/AttributeDescriptorList&gt;     &lt;DataBlock&gt;       &lt;LastBlock Value="00" /&gt;       &lt;BlockNumber Value="00000001" /&gt;       &lt;RawData Value="02093201020304050607" /&gt;     &lt;/DataBlock&gt;   &lt;/SetRequestWithListAndFirstDatablock&gt; &lt;/SetRequest&gt;  // The APDU is 40 bytes. It contains the two attribute // descriptors and 10 bytes of raw-data containing the type and // length of the first data and the first 7 bytes to be written. </pre>	<pre> 0601 07 00 0001 01 091F 02020100020110020932010203040506 070809101112131415161718192021  &lt;WriteRequest&gt;   &lt;ListOfVariableAccessSpecification Qty="0001" &gt;     &lt;WriteDataBlockAccess&gt;       &lt;LastBlock Value="00" /&gt;       &lt;BlockNumber Value="0001" /&gt;     &lt;/WriteDataBlockAccess&gt;   &lt;/ListOfVariableAccessSpecification&gt;   &lt;ListOfData Qty="0001" &gt;     &lt;OctetString Value="02020100020110020932010203040506                                 070809101112131415161718192021" /&gt;   &lt;/ListOfData&gt; &lt;/WriteRequest&gt;  // The APDU is 40 bytes. 31 bytes of octet-string contains raw- // data: the number and the name of objects to be written, the // number of data to be written and the first 21 bytes of the // first data to be written. </pre>
--	--

DLMS/COSEM Architecture and Protocols

C103C1 01 00000003 11 3940414243444546474849500A033030 30  <SetRequest> <SetRequestWithDataBlock> <InvokeIdAndPriority Value="C1" /> <DataBlock> <LastBlock Value="01" /> <BlockNumber Value="00000003" /> <RawData Value="3940414243444546474849500A033030 30" /> </DataBlock> </SetRequestWithDataBlock> </SetRequest>  // The APDU is 26 bytes. 17 bytes of raw-data contain the third // part of the first data and the second data to be written.	0601 07 01 0003 01 0903 303030  <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="01" /> <BlockNumber Value="0003" /> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> <ListOfData Qty="0001" > <OctetString Value="303030" /> </ListOfData> </WriteRequest>  // The APDU is 12 bytes. 3 bytes of octet-string contains raw-data: the value of the second attribute.
C504C1 02 00 00 00000003  <SetResponse> <SetResponseForLastDataBlockWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <_DataAccessResult Value="Success" /> <_DataAccessResult Value="Success" /> </Result> <BlockNumber Value="00000003" /> </SetResponseForLastDataBlockWithList> </SetResponse>	0D02 00 00  <WriteResponse Qty="0002" > <Success /> <Success /> </WriteResponse>

## 14.2 ACCESS service example

Table 148 shows an example of the ACCESS service without general block transfer.

**Table 148 – Example: ACCESS service without block transfer**

Message Elements (MAX APDU = 1024)	Contents	LEN (Bytes)
<b>Access-Request</b>	D9	1
<b>long-invoke-id-and-priority</b>	40000000	4
<b>date-time</b>	00	1
<b>access-request-body</b>		0
<b>access-request-specification</b>		0
<b>SEQUENCE OF CHOICE</b>	04	1
<b>access-request-get</b>	01	1
<b>cosem-attribute-descriptor</b>		0
<b>class-id</b>	0001	2
<b>instance-id</b>	0000600100FF	6
<b>attr-id</b>	02	1
<b>access-request-get</b>	01	1
<b>cosem-attribute-descriptor</b>		0
<b>class-id</b>	0008	2
<b>instance-id</b>	0000010000FF	6
<b>attr-id</b>	02	1
<b>access-request-set</b>	02	1
<b>cosem-attribute-descriptor</b>		0
<b>class-id</b>	0014	2
<b>instance-id</b>	00000D0000FF	6
<b>attr-id</b>	07	1
<b>access-request-set</b>	02	1
<b>cosem-attribute-descriptor</b>		0
<b>class-id</b>	0014	2
<b>instance-id</b>	00000D0000FF	6
<b>attr-id</b>	08	1
<b>access-request-list-of-data</b>		
<b>SEQUENCE OF Data</b>	04	1
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>array</b>	01040203090100090CFFFFFFF 00000000000901F0203090101090CF FFFFFFFFF00000000000901FF0203 090102090CFFFFFFF00000000 000901F0203090103090CFFFFFFF FFFF00000000000901FF	90
<b>array</b>	010402080901001FF11FF11FF11 FF11FF11FF02080901011021101101 1101110111011101020809010211FF11 FF11FF11FF11FF11FF0208090103 1101110211021102110211021102	78

<b>Complete Access-Request APDU (encoded)</b>	D940000000000040100010000600100FF 020100080000010000FF020200140000 0D0000F070200140000D0000FF0804 000001040203090100090CFFFFFFFFF FFFF000000000000901FF020309010109 0CFFFFFFFFF000000000000901FF 0203090102090CFFFFFFFFF0000 0000000901FF0203090103090CFFFFF FFFFFFF000000000000901FF01040208 09010011FF11FF11FF11FF11FF11 FF020809010111021101110111011101 11011101020809010211FF11FF11FF11 FF11FF11FF11FF020809010311011102 11021102110211021102	218
<b>Access-Response</b>	DA	1
<b>long-invoke-id-and-priority</b>	40000000	4
<b>date-time</b>	00	1
<b>access-response-body</b>		0
<b>access-request-specification OPTIONAL</b>	00	1
<b>access-response-list-of-data</b>		0
<b>SEQUENCE OF Data</b>	04	1
<b>octet-string</b>	09083030303030303031	10
<b>octet-string</b>	090C07DC030C07161E0000FF8880	14
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>access-response-specification</b>		0
<b>SEQUENCE OF CHOICE</b>	04	1
<b>access-response-get</b>	01	1
<b>result</b>	00	1
<b>access-response-get</b>	01	1
<b>result</b>	00	1
<b>access-response-set</b>	02	1
<b>result</b>	00	1
<b>access-response-set</b>	02	1
<b>result</b>	00	1
<b>Complete Access-Response APDU (encoded)</b>	DA40000000000000409083030303030 3031090C07DC030C07161E0000FF8880 000004010001000200200	43

### 14.3 Compact array encoding example

#### 14.3.1 General

Any series of data of the same type can be encoded as array or compact-array.

Subclause 14.3 demonstrates how to use compact array encoding to ensure unambiguous results.

#### 14.3.2 The specification of compact-array

Subclause 9.5 specifies the following:

```
Data ::= CHOICE
{
    null-data
        [0] IMPLICIT NULL,
```

```

array [1] IMPLICIT SEQUENCE OF Data,
structure [2] IMPLICIT SEQUENCE OF Data,
boolean [3] IMPLICIT BOOLEAN,
bit-string [4] IMPLICIT BIT STRING,
double-long [5] IMPLICIT Integer32,
double-long-unsigned [6] IMPLICIT Unsigned32,
floating-point [7] IMPLICIT OCTET STRING(SIZE(4)),
octet-string [9] IMPLICIT OCTET STRING,
visible-string [10] IMPLICIT VisibleString,
bcd [13] IMPLICIT Integer8,
utf8-string [12] IMPLICIT NULL,
integer [15] IMPLICIT Integer8,
long [16] IMPLICIT Integer16,
unsigned [17] IMPLICIT Unsigned8,
long-unsigned [18] IMPLICIT Unsigned16,
compact-array [19] IMPLICIT SEQUENCE
{
    contents-description [0] TypeDescription,
    array-contents [1] IMPLICIT OCTET STRING
},
long64 [20] IMPLICIT Integer64,
long64-unsigned [21] IMPLICIT Unsigned64,
enum [22] IMPLICIT Unsigned8,
float32 [23] IMPLICIT OCTET STRING (SIZE(4)),
float64 [24] IMPLICIT OCTET STRING (SIZE(8)),
date_time [25] IMPLICIT OCTET STRING (SIZE(12)),
date [26] IMPLICIT OCTET STRING (SIZE(5)),
time [27] IMPLICIT OCTET STRING (SIZE(4)),
dont-care [255] IMPLICIT NULL
}

-- The following TypeDescription relates to the compact-array data Type

TypeDescription ::= CHOICE
{
    null-data [0] IMPLICIT NULL,
    array [1] IMPLICIT SEQUENCE
    {
        number-of-elements Unsigned16,
        type-description TypeDescription
    },
    structure [2] IMPLICIT SEQUENCE OF TypeDescription,
    boolean [3] IMPLICIT NULL,
    bit-string [4] IMPLICIT NULL,
    double-long [5] IMPLICIT NULL,
    double-long-unsigned [6] IMPLICIT NULL,
    floating-point [7] IMPLICIT NULL,
    octet-string [9] IMPLICIT NULL,
    visible-string [10] IMPLICIT NULL,
    bcd [13] IMPLICIT NULL,
    integer [15] IMPLICIT NULL,
    long [16] IMPLICIT NULL,
    unsigned [17] IMPLICIT NULL,
    long-unsigned [18] IMPLICIT NULL,
    long64 [20] IMPLICIT NULL,
    long64-unsigned [21] IMPLICIT NULL,
    enum [22] IMPLICIT NULL,
    float32 [23] IMPLICIT NULL,
    float64 [24] IMPLICIT NULL,
    date_time [25] IMPLICIT NULL,
    date [26] IMPLICIT NULL,
    time [27] IMPLICIT NULL,
    dont-care [255] IMPLICIT NULL
}

```

Notice that in the compact-array type:

- contents-description / TypeDescription specifies the data type of the elements in the compact array:

- in the case of simple data types it contains the tag of the type. In the case of string types, the length is not part of the TypeDescription: it is conveyed as part of the array contents;

**NOTE** For example if the data includes octet-strings, only the tag [9] is included in the contents-description. The length of the octet-string is included in the array-contents. With this, string type data with different lengths (including a length of 0) can be encoded in the compact-array.

- in the case of *array*, it includes the tag of the array [1], the number of elements in the array (Unsigned16) and the TypeDescription of the elements in the array;
- in the case of *structure* the TypeDescription is specified as a SEQUENCE OF TypeDescription. Therefore, the contents-description includes the tag of the structure [2], the number of elements in the structure and the TypeDescription of each element in the structure.
- the array-contents includes the series of data – when relevant (in the case of string types) together with its length, without repeating the data type – as an octet string;

Note also that although the contents-description and the array-contents elements of the compact-array type are tagged, these tags do not have to be encoded, as specified in IEC 61334-6:2000, 6.9:

*A-XDR encoding of a SEQUENCE value shall be the A-XDR encoding of one data value from each of the types listed in the ASN.1 definition of the SEQUENCE type, in the order of their appearance in the definition, unless the type was referenced with the keyword "OPTIONAL" or the keyword "DEFAULT".*

*Tags of explicitly tagged components of a SEQUENCE value represent redundant information, therefore are not encoded: A-XDR encoding of an explicit tagged component value is the A-XDR encoding of the component value.*

#### 14.3.3 Example 1: Compact array encoding an array of five long-unsigned values

An array of 5 elements of type long-unsigned has to be encoded.

The values of the five elements are: 11 11, 22 22, 33 33, 44 44, 55 55.

Encoding as array	Encoding as compact-array
01 // tag of array	13 // tag of compact-array
05 // number of elements	// contents-description
12 11 11 // tag of long-unsigned type and first value	12 // tag of the long-unsigned type
12 22 22 // tag of long-unsigned type and second value	// array-contents
12 33 33 // etc.	0A // length of octet-string
12 44 44	11 11 22 22 33 33 44 44 55 55
12 55 55	// the five values

The length of the encoded data in the two cases is shown in the table below. In the case of the long-unsigned type, 33% per element can be saved.

	Array	Compact array	Gain compared to array
Header	2	3	-1
First element	3	2	1
Second element	3	2	1
Third element	3	2	1
Fourth element	3	2	1
Fifth element	3	2	1
<b>Total</b>	<b>17</b>	<b>13</b>	<b>4</b>

#### 14.3.4 Example 2: Compact-array encoding of five octet-string values

An array of five octet-string values has to be encoded, of which one is of zero length.

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	551/614
-----------------------	------------	-----------------------	---------

- 31 32 33 34 35 36 37 38
- 41 42 43 44 45 46 47 48
- -
- 31 32 33 34 35 36 37 38
- 41 42 43 44 45 46 47 48
- 

Encoding as array	Encoding as compact-array
01 // tag of array 05 // number of elements  09 08 31 32 33 34 35 36 37 38 // type – length - value 09 08 41 42 43 44 45 46 47 48 // second value 09 00 // third value, octet-string of length 0 09 08 31 32 33 34 35 36 37 38 // fourth value 09 08 41 42 43 44 45 46 47 48 // fifth value	13 // tag of compact array // contents description 09 // tag for octet-string // array contents 25 // length of octet-string, 37 bytes 08 31 32 33 34 35 36 37 38 // length - value 08 41 42 43 44 45 46 47 48 // second length - value 00 // third value octet-string of length 0 08 31 32 33 34 35 36 37 38 // fourth length - value 08 41 42 43 44 45 46 47 48 // fifth length - value

The length of the encoded data in the two cases is shown in the table below.

In the case of octet-string, the gain depends on the length of the octet-string. In the case of octet-string of length zero (null-data) the gain is 50% per element.

	Array	Compact array	Gain compared to array
Header	2	3	-1
First element	10	9	1
Second element	10	9	1
Third element	2	1	1
Fourth element	10	9	1
Fifth element	10	9	1
<b>Total</b>	<b>44</b>	<b>40</b>	<b>4</b>

#### 14.3.5 Example 3: Encoding of the buffer of a Profile generic object

The profile has a time stamp column, a status column, and two columns carrying a double-long-unsigned value each (e.g. A+ and A-, import and export active energy). The capture period is 900 s. There are 96 entries (one day).

NOTE If instead of register readings just delta values would be stored, they could be represented as long-unsigned instead of double-long-unsigned.

Entry	Timestamp	Status	Value	Value	Bytes
1	07D00101FF000000FF800000	80	00000101	00000001	21
2	07D00101FF000F00FF800000	00	00000102	00000002	21
3	07D00101FF001E00FF800000	00	00000103	00000003	21
4	07D00101FF002D00FF800000	00	00000104	00000004	21
....					...
96	07D00101FF172D00FF800000	00	00000196	00000096	21
				Total bytes	2 016

Encoding as array (using the null-data feature)	Encoding as compact-array
01 60 // array of 96 elements	13 // compact-array // contents-description 02 04 09110606 // structure of four elements: // octet-string, // unsigned, // double-long-unsigned, // double-long-unsigned
	// array-contents 8203CC // length of octet-string 972 bytes
// first structure, 28 bytes 0204 // structure of 4 elements 090C07D00101FF000000FF800000 1180 // status value is 80 0600000101 0600000001	// first structure, 22 bytes 0C07D00101FF000000FF800000 80 00000101 00000001
// second structure, 15 bytes 0204 00 // null-data for time stamp 1100 // status value is 0 0600000102 0600000002	// second structure, 10 bytes 00 // an octet-string of length 0 has the same effect as null-data 00 // status value is 0 00000102 00000002
// third structure 14 bytes 0204 00 // null-data for time stamp 00 // null-data for status 0600000103 0600000003	// third structure, 10 bytes 00 // octet-string of length 0 00 // status value is 0 00000103 00000003
// fourth structure 14 bytes 0204 00 // null-data for time stamp 00 // null-data for status 0600000104 0600000004	// fourth structure, 10 bytes 00 // octet-string of length 0 00 // status value is 0 00000104 00000004
...	...
// ninety-sixth structure 14 bytes 0204 00 // null-data for time stamp 00 // null-data for status 0600000196 0600000096	// ninety-sixth structure, 10 bytes 00 // octet-string of length 0 00 // status value is 0 00000196 00000096
In the case of using compact array, it is not possible to know the number of elements from the length of the octet string of the array-contents. In the case of compact array encoding, the null-data feature can be applied only for string type data.	

The length of the encoded data in the two cases is shown in the table below.

	Array (using null-data)	Compact array	Gain compared to array
Header	2	10	-8
First structure	28	22	6
Second structure	15	10	5
Third structure	14	10	4
Fourth structure	14	10	4
96th structure	14	10	4
<b>Total</b>	<b>1361</b>	<b>982</b>	<b>375</b>
<b>Encoded data in % of raw data</b>	<b>68%</b>	<b>49%</b>	

When encoding the data as an array, the use of the null data feature allows compression of 32% in this example. When encoding as a compact array the compression is 51%.

## 14.4 Profile generic IC buffer attribute encoding examples

### 14.4.1 General

The following tables show encoding examples of reading a Profile generic IC buffer attribute with:

- normal encoding;
- null-data encoding;
- compact-array encoding;
- null-data and delta-value encoding combined;
- the various methods combined with compression.

### 14.4.2 Get-response with Profile generic normal encoding example

Table 149 shows an encoding example for a get-response-normal APDU reading a Profile generic object buffer attribute with normal encoding. The content of the buffer is determined by the capture\_objects attribute that references:

- the Clock object time attribute;
- the value attribute of a status object; and
- the value attribute of an energy register.

By selective access parameters, one day is selected with 24 entries. The size of the APDU is 558 bytes.

**Table 149 – Profile generic buffer – get-response with normal encoding**

Message Elements	Index	Value	Contents	LEN (Bytes)
<b>get-response</b>			C4	1
<b>get-response-normal</b>			01	1
<b>invoke-id-and-priority</b>			00	1
<b>result CHOICE</b>				0
<b>data</b>			00	1
<b>array [24]</b>			0118	2
<b>structure [3]</b>	0		0203	2
<b>octet-string</b>		12-2-2018 0:00:00	090C07E2020C05000 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		100000	06000186A0	5
<b>structure [3]</b>	1		0203	2
<b>octet-string</b>		12-2-2018 1:00:00	090C07E2020C05010 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		100416	0600018840	5
<b>structure [3]</b>	2		0203	2
<b>octet-string</b>		12-2-2018 2:00:00	090C07E2020C05020 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		100832	06000189E0	5
<b>structure [3]</b>	3		0203	2
<b>octet-string</b>		12-2-2018 3:00:00	090C07E2020C05030 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		101248	0600018B80	5
<b>structure [3]</b>	4		0203	2

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	555/614
-----------------------	------------	-----------------------	---------

Message Elements	Index	Value	Contents	LEN (Bytes)
<b>octet-string</b>		12-2-2018 4:00:00	090C07E2020C05040 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		101664	0600018D20	5
<b>structure [3]</b>	5		0203	2
<b>octet-string</b>		12-2-2018 5:00:00	090C07E2020C05050 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		102080	0600018EC0	5
<b>structure [3]</b>	6		0203	2
<b>octet-string</b>		12-2-2018 6:00:00	090C07E2020C05060 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		102496	0600019060	5
<b>structure [3]</b>	7		0203	2
<b>octet-string</b>		12-2-2018 7:00:00	090C07E2020C05070 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		102912	0600019200	5
<b>structure [3]</b>	8		0203	2
<b>octet-string</b>		12-2-2018 8:00:00	090C07E2020C05080 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		103328	06000193A0	5
<b>structure [3]</b>	9		0203	2
<b>octet-string</b>		12-2-2018 9:00:00	090C07E2020C05090 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		103744	0600019540	5
<b>structure [3]</b>	10		0203	2
<b>octet-string</b>		12-2-2018 10:00:00	090C07E2020C050A0 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		104160	06000196E0	5
<b>structure [3]</b>	11		0203	2
<b>octet-string</b>		12-2-2018 11:00:00	090C07E2020C050B0 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		104576	0600019880	5
<b>structure [3]</b>	12		0203	2
<b>octet-string</b>		12-2-2018 12:00:00	090C07E2020C050C0 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		104992	0600019A20	5
<b>structure [3]</b>	13		0203	2
<b>octet-string</b>		12-2-2018 13:00:00	090C07E2020C050D0 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		105408	0600019BC0	5
<b>structure [3]</b>	14		0203	2
<b>octet-string</b>		12-2-2018 14:00:00	090C07E2020C050E0 00000800000	14

Message Elements	Index	Value	Contents	LEN (Bytes)
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		105824	0600019D60	5
<b>structure [3]</b>	15		0203	2
<b>octet-string</b>		12-2-2018 15:00:00	090C07E2020C050F0 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		106240	0600019F00	5
<b>structure [3]</b>	16		0203	2
<b>octet-string</b>		12-2-2018 16:00:00	090C07E2020C05100 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		106656	060001A0A0	5
<b>structure [3]</b>	17		0203	2
<b>octet-string</b>		12-2-2018 17:00:00	090C07E2020C05110 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		107072	060001A240	5
<b>structure [3]</b>	18		0203	2
<b>octet-string</b>		12-2-2018 18:00:00	090C07E2020C05120 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		107488	060001A3E0	5
<b>structure [3]</b>	19		0203	2
<b>octet-string</b>		12-2-2018 19:00:00	090C07E2020C05130 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		107904	060001A580	5
<b>structure [3]</b>	20		0203	2
<b>octet-string</b>		12-2-2018 20:00:00	090C07E2020C05140 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		108320	060001A720	5
<b>structure [3]</b>	21		0203	2
<b>octet-string</b>		12-2-2018 21:00:00	090C07E2020C05150 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		108736	060001A8C0	5
<b>structure [3]</b>	22		0203	2
<b>octet-string</b>		12-2-2018 22:00:00	090C07E2020C05160 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		109152	060001AA60	5
<b>structure [3]</b>	23		0203	2
<b>octet-string</b>		12-2-2018 23:00:00	090C07E2020C05170 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		109568	060001AC00	5

Message Elements	Index	Value	Contents	LEN (Bytes)
The complete get-response APDU:  C401000001180203090C07E2020C0500000000800000110006000186A00203090C07E2020C050100000800000110006000188400203090C07E2020C0502000000800000110006000189E00203090C07E2020C0503000000800000110006000018B800203090C07E2020C050400000080000011000600018D200203090C07E2020C0505000000800000110006000118EC00203090C07E2020C0506000000800000110006000190600203090C07E2020C0507000000800000110006000192000203090C07E2020C0508000000800000110006000193A00203090C07E2020C0509000000800000110006000195400203090C07E2020C050A000000800000110006000196E00203090C07E2020C050B00000080000110006000198800203090C07E2020C050C00000080000011000600019A200203090C07E2020C050D00000080000011000600019BC00203090C07E2020C050E000000080000011000600019D600203090C07E2020C050F000000080000011000600019F000203090C07E2020C05100000008000001100060001A0A00203090C07E2020C051100000008000001100060001A2400203090C07E2020C051200000008000001100060001A3E00203090C07E2020C051300000008000001100060001A5800203090C07E2020C051400000008000001100060001A7200203090C07E2020C05150000000800000110060001A8C00203090C07E2020C051600000008000001100060001AA600203090C07E2020C0517000008000001100060001AC00		558		

#### 14.4.3 Get-response with Profile generic null-data compressed encoding example

Table 150 shows an encoding example for the same get-response APDU but using null-data encoding. The time stamp is provided for the first reading only. For the subsequent entries null-data are provided since from the capture\_period attribute it is known that the capture occurs hourly. Similarly, the status is provided only in the first entry because it is the same for each subsequent entry. This method results in a buffer size of 236 bytes compared to 558 for normal encoding.

**Table 150 – Profile generic buffer – get-response with null-data compression**

Message Elements	Index	Value	Contents	LEN (Bytes)
<b>get-response</b>			C4	1
<b>get-response-normal</b>			01	1
<b>invoke-id-and-priority</b>			00	1
<b>result CHOICE</b>				0
<b>data</b>			00	1
<b>array [24]</b>			0118	2
<b>structure [3]</b>	0		0203	2
<b>octet-string</b>		12-2-2018 0:00:00	090C07E2020C05000 00000800000	14
<b>unsigned</b>		0	1100	2
<b>double-long-unsigned</b>		100000	06000186A0	5
<b>structure [3]</b>	1		0203	2
<b>null-data</b>		12-2-2018 1:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		100416	0600018840	5
<b>structure [3]</b>	2		0203	2
<b>null-data</b>		12-2-2018 2:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		100832	06000189E0	5
<b>structure [3]</b>	3		0203	2
<b>null-data</b>		12-2-2018 3:00:00	00	1
<b>null-data</b>		0	00	1

558/614	2021-12-22	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

Message Elements	Index	Value	Contents	LEN (Bytes)
<b>double-long-unsigned</b>		101248	0600018B80	5
<b>structure [3]</b>	4		0203	2
<b>null-data</b>		12-2-2018 4:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		101664	0600018D20	5
<b>structure [3]</b>	5		0203	2
<b>null-data</b>		12-2-2018 5:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		102080	0600018EC0	5
<b>structure [3]</b>	6		0203	2
<b>null-data</b>		12-2-2018 6:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		102496	0600019060	5
<b>structure [3]</b>	7		0203	2
<b>null-data</b>		12-2-2018 7:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		102912	0600019200	5
<b>structure [3]</b>	8		0203	2
<b>null-data</b>		12-2-2018 8:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		103328	06000193A0	5
<b>structure [3]</b>	9		0203	2
<b>null-data</b>		12-2-2018 9:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		103744	0600019540	5
<b>structure [3]</b>	10		0203	2
<b>null-data</b>		12-2-2018 10:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		104160	06000196E0	5
<b>structure [3]</b>	11		0203	2
<b>null-data</b>		12-2-2018 11:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		104576	0600019880	5
<b>structure [3]</b>	12		0203	2
<b>null-data</b>		12-2-2018 12:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		104992	0600019A20	5
<b>structure [3]</b>	13		0203	2
<b>null-data</b>		12-2-2018 13:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		105408	0600019BC0	5

Message Elements	Index	Value	Contents	LEN (Bytes)
<b>structure [3]</b>	14		0203	2
<b>null-data</b>		12-2-2018 14:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		105824	0600019D60	5
<b>structure [3]</b>	15		0203	2
<b>null-data</b>		12-2-2018 15:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		106240	0600019F00	5
<b>structure [3]</b>	16		0203	2
<b>null-data</b>		12-2-2018 16:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		106656	060001A0A0	5
<b>structure [3]</b>	17		0203	2
<b>null-data</b>		12-2-2018 17:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		107072	060001A240	5
<b>structure [3]</b>	18		0203	2
<b>null-data</b>		12-2-2018 18:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		107488	060001A3E0	5
<b>structure [3]</b>	19		0203	2
<b>null-data</b>		12-2-2018 19:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		107904	060001A580	5
<b>structure [3]</b>	20		0203	2
<b>null-data</b>		12-2-2018 20:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		108320	060001A720	5
<b>structure [3]</b>	21		0203	2
<b>null-data</b>		12-2-2018 21:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		108736	060001A8C0	5
<b>structure [3]</b>	22		0203	2
<b>null-data</b>		12-2-2018 22:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		109152	060001AA60	5
<b>structure [3]</b>	23		0203	2
<b>null-data</b>		12-2-2018 23:00:00	00	1
<b>null-data</b>		0	00	1
<b>double-long-unsigned</b>		109568	060001AC00	5

Message Elements	Index	Value	Contents	LEN (Bytes)
The complete get-response APDU:  C401000001180203090C07E2020C0500000000800000110006000186A00203000060001884002 0300006000189E002030000600018B8002030000600018D2002030000600018EC002030000 060001906002030000600019200020300006000193A0020300006000195400203000060001 96E00203000060001988002030000600019A20002030000600019BC002030000600019D6002 030000600019F000203000060001A0A00203000060001A2400203000060001A3E002030000 060001A5800203000060001A7200203000060001A8C00203000060001AA600203000060001 AC00				236

#### 14.4.4 Get-response with Profile generic compact-array encoding example

Table 151 shows an encoding example for the same get-response APDU but using compact-array encoding. This method results in a buffer size of 168 bytes compared to 558 for normal encoding.

**Table 151 – Profile generic buffer – get-response with compact-array encoding**

Message Elements	Index	Value	Contents	LEN (Bytes)
<b>get-response</b>			C4	1
<b>get-response-normal</b>			01	1
<b>invoke-id-and-priority</b>			00	1
<b>result CHOICE</b>				0
<b>data</b>			00	1
<b>compact-array</b>			13	1
<b>contents-description</b>				0
<b>structure [3]</b>			0203	2
<b>octet-string</b>			09	1
<b>unsigned</b>			11	1
<b>double-long-unsigned</b>			06	1
<b>array-contents</b>		156	819C	2
<b>structure [3]</b>	0			0
<b>octet-string</b>		12-2-2018 0:00:00	0C07E2020C0500 000000800000	13
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		100000	000186A0	4
<b>structure [3]</b>	1			0
<b>octet-string</b>		12-2-2018 1:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		100416	00018840	4
<b>structure [3]</b>	2			0
<b>octet-string</b>		12-2-2018 2:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		100832	000189E0	4
<b>structure [3]</b>	3			0
<b>octet-string</b>		12-2-2018 3:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		101248	00018B80	4

Message Elements	Index	Value	Contents	LEN (Bytes)
<b>structure [3]</b>	4			0
<b>octet-string</b>		12-2-2018 4:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		101664	00018D20	4
<b>structure [3]</b>	5			0
<b>octet-string</b>		12-2-2018 5:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		102080	00018EC0	4
<b>structure [3]</b>	6			0
<b>octet-string</b>		12-2-2018 6:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		102496	00019060	4
<b>structure [3]</b>	7			0
<b>octet-string</b>		12-2-2018 7:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		102912	00019200	4
<b>structure [3]</b>	8			0
<b>octet-string</b>		12-2-2018 8:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		103328	000193A0	4
<b>structure [3]</b>	9			0
<b>octet-string</b>		12-2-2018 9:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		103744	00019540	4
<b>structure [3]</b>	10			0
<b>octet-string</b>		12-2-2018 10:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		104160	000196E0	4
<b>structure [3]</b>	11			0
<b>octet-string</b>		12-2-2018 11:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		104576	00019880	4
<b>structure [3]</b>	12			0
<b>octet-string</b>		12-2-2018 12:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		104992	00019A20	4
<b>structure [3]</b>	13			0
<b>octet-string</b>		12-2-2018 13:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		105408	00019BC0	4
<b>structure [3]</b>	14			0

Message Elements	Index	Value	Contents	LEN (Bytes)
<b>octet-string</b>		12-2-2018 14:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		105824	00019D60	4
<b>structure [3]</b>	15			0
<b>octet-string</b>		12-2-2018 15:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		106240	00019F00	4
<b>structure [3]</b>	16			0
<b>octet-string</b>		12-2-2018 16:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		106656	0001A0AO	4
<b>structure [3]</b>	17			0
<b>octet-string</b>		12-2-2018 17:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		107072	0001A240	4
<b>structure [3]</b>	18			0
<b>octet-string</b>		12-2-2018 18:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		107488	0001A3E0	4
<b>structure [3]</b>	19			0
<b>octet-string</b>		12-2-2018 19:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		107904	0001A580	4
<b>structure [3]</b>	20			0
<b>octet-string</b>		12-2-2018 20:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		108320	0001A720	4
<b>structure [3]</b>	21			0
<b>octet-string</b>		12-2-2018 21:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		108736	0001A8C0	4
<b>structure [3]</b>	22			0
<b>octet-string</b>		12-2-2018 22:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		109152	0001AA60	4
<b>structure [3]</b>	23			0
<b>octet-string</b>		12-2-2018 23:00:00	00	1
<b>unsigned</b>		0	00	1
<b>double-long-unsigned</b>		109568	0001AC00	4

Message Elements	Index	Value	Contents	LEN (Bytes)
The complete get-response APDU:  C4010000130203091106819C0C07E2020C0500000000800000000000186A000000001884000000 0189E0000000018B80000000018D20000000018EC00000000190600000000192000000000193A0 0000000195400000000196E0000000019880000000019A20000000019BC0000000019D6000000 019F0000000001A0A000000001A24000000001A3E000000001A58000000001A72000000001A8C0 00000001AA6000000001AC00				168

#### 14.4.5 Get-response with Profile generic null-data and delta-value encoding example

Table 152 shows an encoding example for the same get-response APDU but using null-data encoding combined with delta-value encoding. The first entry contains the full time stamp, the status and the full register value. For subsequent entries, null-data are provided for the time stamp and the status and delta-value is provided for the register value. In this example the change is a constant value of 41 which can be encoded as a delta--unsigned instead of a double-long-unsigned. This method results in a buffer size of 167 bytes compared to 558 for normal encoding.

**Table 152 – Profile generic buffer – Get-response with null-data and delta-value encoding**

Message Elements	Index	Value	Delta	Contents	LEN (Bytes)
<b>get-response</b>				C4	1
<b>get-response-normal</b>				01	1
<b>invoke-id-and-priority</b>				00	1
<b>result CHOICE</b>					0
<b>data</b>				00	1
<b>array [24]</b>				0118	2
<b>structure [3]</b>	0			0203	2
<b>octet-string</b>		12-2-2018 0:00:00		090C07E2020 C0500000000 800000	14
<b>unsigned</b>		0		1100	2
<b>double-long-unsigned</b>		100000		06000186A0	5
<b>structure [3]</b>	1	0		0203	2
<b>null-data</b>		12-2-2018 1:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100041	41	1F29	2
<b>structure [3]</b>	2			0203	2
<b>null-data</b>		12-2-2018 2:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100082	41	1F29	2
<b>structure [3]</b>	3			0203	2
<b>null-data</b>		12-2-2018 3:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100123	41	1F29	2
<b>structure [3]</b>	4			0203	2
<b>null-data</b>		12-2-2018 4:00:00		00	1
<b>null-data</b>		0		00	1

564/614	2021-12-22	DLMS UA 1000-2 Ed. 11	DLMS User Association
---------	------------	-----------------------	-----------------------

Message Elements	Index	Value	Delta	Contents	LEN (Bytes)
<b>unsigned-delta</b>		100164	41	1F29	2
<b>structure [3]</b>	5			0203	2
<b>null-data</b>		12-2-2018 5:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100205	41	1F29	2
<b>structure [3]</b>	6			0203	2
<b>null-data</b>		12-2-2018 6:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100246	41	1F29	2
<b>structure [3]</b>	7			0203	2
<b>null-data</b>		12-2-2018 7:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100287	41	1F29	2
<b>structure [3]</b>	8			0203	2
<b>null-data</b>		12-2-2018 8:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100328	41	1F29	2
<b>structure [3]</b>	9			0203	2
<b>null-data</b>		12-2-2018 9:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100369	41	1F29	2
<b>structure [3]</b>	10			0203	2
<b>null-data</b>		12-2-2018 10:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100410	41	1F29	2
<b>structure [3]</b>	11			0203	2
<b>null-data</b>		12-2-2018 11:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100451	41	1F29	2
<b>structure [3]</b>	12			0203	2
<b>null-data</b>		12-2-2018 12:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100492	41	1F29	2
<b>structure [3]</b>	13			0203	2
<b>null-data</b>		12-2-2018 13:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100533	41	1F29	2
<b>structure [3]</b>	14			0203	2
<b>null-data</b>		12-2-2018 14:00:00		00	1
<b>null-data</b>		0		00	1
<b>unsigned-delta</b>		100574	41	1F29	2
<b>structure [3]</b>	15			0203	2

Message Elements	Index	Value	Delta	Contents	LEN (Bytes)
null-data		12-2-2018 15:00:00		00	1
null-data		0		00	1
unsigned-delta		100615	41	1F29	2
structure [3]	16			0203	2
null-data		12-2-2018 16:00:00		00	1
null-data		0		00	1
unsigned-delta		100656	41	1F29	2
structure [3]	17			0203	2
null-data		12-2-2018 17:00:00		00	1
null-data		0		00	1
unsigned-delta		100697	41	1F29	2
structure [3]	18			0203	2
null-data		12-2-2018 18:00:00		00	1
null-data		0		00	1
unsigned-delta		100738	41	1F29	2
structure [3]	19			0203	2
null-data		12-2-2018 19:00:00		00	1
null-data		0		00	1
unsigned-delta		100779	41	1F29	2
structure [3]	20			0203	2
null-data		12-2-2018 20:00:00		00	1
null-data		0		00	1
unsigned-delta		100820	41	1F29	2
structure [3]	21			0203	2
null-data		12-2-2018 21:00:00		00	1
null-data		0		00	1
unsigned-delta		100861	41	1F29	2
structure [3]	22			0203	2
null-data		12-2-2018 22:00:00		00	1
null-data		0		00	1
unsigned-delta		100902	41	1F29	2
structure [3]	23			0203	2
null-data		12-2-2018 23:00:00		00	1
null-data		0		00	1
unsigned-delta		100943	41	1F29	2
The complete get-response APDU:					167
C401000001180203090C07E2020C0500000000800000110006000186A0020300001F2902030000 1F29020300001F29020300001F29020300001F29020300001F29020300001F29020300001F2902 0300001F29020300001F29020300001F29020300001F29020300001F29020300001F2902030000 1F29020300001F29020300001F29020300001F29020300001F29020300001F29020300001F2902 0300001F29020300001F29					

#### 14.4.6 Comparison of various encoding methods for Get-response APDU

Table 153 summarizes the result of the various encoding methods presented, also showing the results when data for 2, 4 or 7 days are captured.

#### 14.4.7 Combination of the various encoding methods and V.44 compression

V.44 compression can be used to further decrease the size of the APDU. Table 154 shows the resulting APDU sizes for the same examples shown in Table 153.

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	567/614
-----------------------	------------	-----------------------	---------

**Table 153 – Comparison of various encoding methods for get-response APDU**

Profile Generic Parameters				Profile Generic Encoding									
Records	Period	Interval	Values	Normal		Compressed null-data		compact-array		Compressed null-data & delta values		Compressed null-data & delta values (const)	
Number	Minutes	Days	Number	Size	Ratio	Size	Ratio	Size	Ratio	Size	Ratio	Size	Ratio
24	60	1	1	558	100%	236	42%	168	30%	168	30%	167	30%
48	60	2	1	1.110	100%	452	41%	313	28%	336	30%	311	28%
96	60	4	1	2.214	100%	884	40%	601	27%	669	30%	599	27%
168	60	7	1	3.871	100%	1.533	40%	1.033	27%	1.177	30%	1.032	27%

**Table 154 – Combination of the various encoding methods and V.44 compression for get-response APDU**

Profile Generic Parameters				Uncompressed	Profile Generic Compression with V.44									
Records	Period	Interval	Values	Normal	Normal		Compressed null-data		compact-array		Compressed null-data & delta values		Compressed null-data & delta values (const)	
Number	Minutes	Days	Number	Size	Size	Ratio	Size	Ratio	Size	Ratio	Size	Ratio	Size	Ratio
24	60	1	1	558	164	29%	111	20%	105	19%	82	15%	34	6%
48	60	2	1	1110	329	30%	193	17%	190	17%	137	12%	33	3%
96	60	4	1	2214	560	25%	377	17%	352	16%	255	12%	34	2%
168	60	7	1	3871	1.146	30%	676	17%	649	17%	458	12%	34	1%

## 15 Data transfer service examples over LPWAN using LoRaWAN Technology

### **15.1 Example of DLMS/COSEM GET service transported through LPWAN using LoRaWAN technology**

**Table 155 – Get service example**

Message Elements	Contents	LEN (Bytes)
<b>get-request</b>	C0	1
<b>get-request-normal</b>	01	1
<b>invoke-id-and-priority</b>	40	1
<b>cosem-attribute-descriptor</b>		0
<b>class-id</b>	0001	2
<b>instance-id</b>	0000600100FF	6
<b>attr-id</b>	02	1
<b>access-selection</b>	00	1
<b>get-request (encoded)</b>	C0014000010000600100FF0200	13
<b>IPv6</b>		
<b>IPv6 Header</b>	60045604001E1180	8
<b>IPv6 Source Address</b>	FE80000000000000745500FFE000100	16
<b>IPv6 Destination Address</b>	FE80000000000000745500FFE000101	16
<b>UDP</b>		
<b>UDP Header</b>	0FDB0FDB001E3A86	8
<b>COSEM Wrapper</b>		
<b>COSEM Wrapper Header</b>	000100010001000D	8
<b>get-request</b>	C0014000010000600100FF0200	13
<b>IPv6 (encoded)</b>	60045604001E1180FE80000000000000745500FFE000100FE80000000000000745500FFE0001010001000DC0014000010000600100FF0200	69
<b>SCHC Packet</b>		
<b>Rule ID=0x02 CompressionResidue= SCHC Payload   Padding=</b>	02C0014000010000600100FF0200	14
<b>SCHC (encoded)</b>	02C0014000010000600100FF0200	14
<b>LoRaWAN (Downlink)</b>		
<b>LoRaWAN Header (Mtype=011b,RFU=000b,Major=00)</b>	60	1
<b>LoRaWAN FHDR (LoRaWAN FHDR (DevAddr=260116CDh,FCtrl=10000000b,FCnt=0001,FOpts=)</b>	CD160126800100	7
<b>LoRaWAN Fport</b>	02	1
<b>LoRaWAN Payload (Encrypted SCHC= C0014000010000600100FF0200) AppSKey = 0x8F282F3F9E901CDC37C3311F01050B2</b>	2F4D36ED10CEEDD7B588C9CA01	13
<b>MIC (little Indian) NwkSKey = 0x8A69FAD35B3CF856EC86EFE77B3F21C4</b>	A1CD20D1	4
<b>LoRaWAN (encoded)</b>	60CD160126800100022F4D36ED10CEEDD7B588C9CA01A1CD20D1	26

Message Elements	Contents	LEN (Bytes)
<b>get-response</b>	C4	1
<b>get-response-normal</b>	01	1
<b>invoke-id-and-priority</b>	40	1
<b>result CHOICE</b>		0
<b>data</b>	00	1
<b>octet-string</b>	09083030303030303031	10
<b>get-response (encoded)</b>	C401400009083030303030303031	14
<b>IPv6</b>		
<b>IPv6 Header</b>	60045604001E1180	8
<b>IPv6 Source Address</b>	FE80000000000000745500FFFE000101	16
<b>IPv6 Destination Address</b>	FE80000000000000745500FFFE000100	16
<b>UDP</b>		
<b>UDP Header</b>	0FDB0FDB001E3A86	8
<b>COSEM Wrapper</b>		
<b>COSEM Wrapper Header</b>	000100010001000E	8
<b>get-response</b>	C4014000090830303030303031	14
<b>IPv6 (encoded)</b>	60045604001E1180FE80000000000000745500FFFE000101FE80000000000000745500FFFE0001000FFDB0FDB001E3A8600010001000EC40140000908303030303031	70
<b>SCHC Packet</b>		
<b>Rule ID=0x02 CompressionResidue= SCHC Payload   Padding=</b>	02C4014000090830303030303031	15
<b>SCHC (encoded)</b>	02C4014000090830303030303031	15
<b>LoRaWAN (Uplink)</b>		
<b>LoRaWAN Header (Mtype=010b,RFU=000b,Major=00b)</b>	40	1
<b>LoRaWAN FHDR (DevAddr=260116CDh,FCtrl=10000000b,FCnt=0001,FOpts=)</b>	CD160126800100	7
<b>LoRaWAN Fport (= RuleID)</b>	02	1
<b>LoRaWAN Payload (Encrypted SCHC=C40140000908303030303031) AppSKey = 0x88F282F3F9E901CDC37C3311F01050B2</b>	5FFABD9D2ADDAF75E0B1A0DC65D1	14
<b>MIC (little endian) NwkSKey = 0x8A69FAD35B3CF856EC86EFE77B3F21C4</b>	E07194EE	4
<b>LoRaWAN (encoded)</b>	40CD160126800100025FFABD9D2ADDAF75E0B1A0DC65D1E07194EE	27

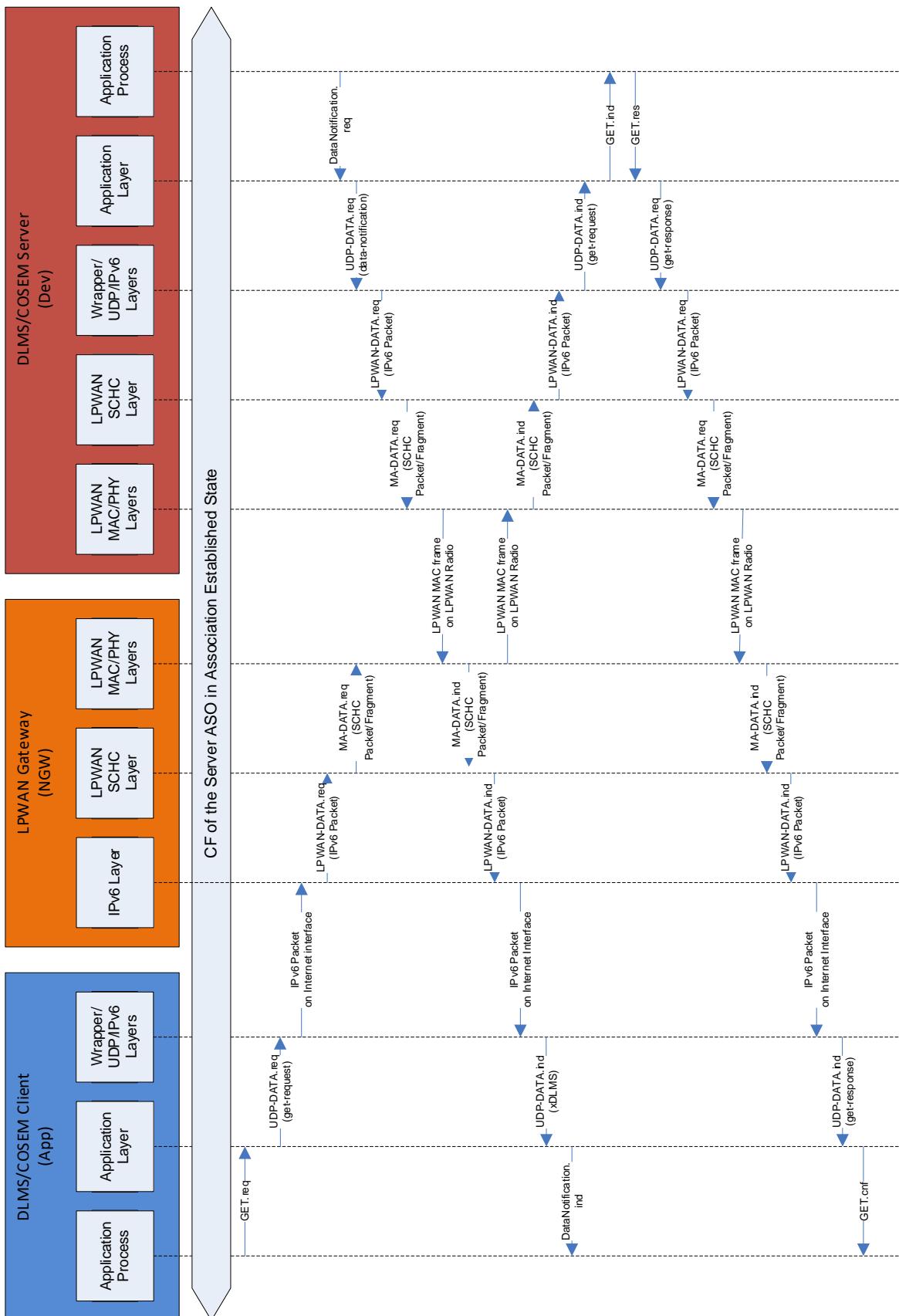


Figure 203 – The DLMS/COSEM GET service on LPWAN

## 15.2 Example of DLMS/COSEM DataNotification service transported through LPWAN with SCHC Fragments

**Table 156 – Data-Notification service with Profile generic**

Message Elements	Contents	LEN (Bytes)
<b>data-notification</b>	0F	1
<b>long-invoke-id-and-priority</b>	00000001	4
<b>date-time</b>	00	1
<b>notification-body</b>		0
<b>data-value</b>		0
<b>array [24]</b>	0118	2
<b>structure [3]</b>	0203	2
<b>octet-string</b>	090C07E2020C0500000000800000	14
<b>unsigned</b>	1100	2
<b>double-long-unsigned</b>	06000186A0	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600018840	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	06000189E0	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600018B80	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600018D20	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600018EC0	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600019060	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600019200	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	06000193A0	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600019540	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	06000196E0	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600019880	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600019A20	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1

Message Elements	Contents	LEN (Bytes)
<b>double-long-unsigned</b>	0600019BC0	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600019D60	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	0600019F00	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	060001A0A0	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	060001A240	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	060001A3E0	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	060001A580	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	060001A720	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	060001A8C0	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	060001AA60	5
<b>structure [3]</b>	0203	2
<b>null-data</b>	00	1
<b>null-data</b>	00	1
<b>double-long-unsigned</b>	060001AC00	5
	0F000000010001180203090C07E20 20C050000000800001100060001 86A0020300006000188400203000 006000189E002030000600018B80 02030000600018D200203000060 0018EC0020300006000190600203 0000060001920002030000600019 3A00203000060001954002030000 06000196E0020300006000198800 20300000600019A2002030000600 019BC002030000600019D6002030 0000600019F000203000060001A0 A00203000060001A240020300000 60001A3E00203000060001A58002 030000060001A720020300006000 1A8C00203000060001AA60020300 00060001AC00	238
<b>IPv6</b>		
<b>IPv6 Header</b>	60045604001E1180	8
<b>IPv6 Source Address</b>	FE80000000000000745500FFFE000101	16
<b>IPv6 Destination Address</b>	FE80000000000000745500FFFE000100	16
<b>UDP</b>		
<b>UDP Header</b>	0FDB0FDB001E3A86	8
<b>COSEM Wrapper</b>		
<b>COSEM Wrapper Header</b>	00010001000100EE	8

Message Elements	Contents	LEN (Bytes)
<b>IPv6 (encoded)</b>	60045604001E1180FE8000000000000000 745500FFE000101FE8000000000000000 745500FFE0001000FDB0FDB001E3A86 00010001000100EE0E000000000000118 0203090C07E2020C0500000000800000 110006000186A0020300000600018840 0203000006000189E002030000060001 8B80020300000600018D200203000006 00018EC0020300000600019060020300 0006000192000203000006000193A002 03000006000195400203000006000196 E0020300000600019880020300000600 019A20020300000600019BC002030000 0600019D60020300000600019F000203 0000060001A0A002030000060001A240 02030000060001A3E002030000060001 A58002030000060001A7200203000006 0001A8C002030000060001AA60020300 00060001AC00	294
<b>SCHC</b>		
<b>C/D</b>		
<b>SCHC Header (Rule ID=09, Compression Residue=)   SCHC Payload</b>	090E0000000000001180203090C07E202 0C05000000008000011006000186A0 02030000060001884002030000060001 89E0020300000600018B800203000006 00018D20020300000600018EC0020300 00060001906002030000060001920002 03000006000193A00203000006000195 400203000006000196E0020300000600 019880020300000600019A2002030000 0600019BC0020300000600019D600203 00000600019F0002030000060001A0A0 02030000060001A24002030000060001 A3E002030000060001A5800203000006 0001A72002030000060001A8C0020300 00060001AA6002030000060001AC00	239
<b>F/R</b>		
<b>SCHC Fragment 1 (Uplink), Current LoRaWAN MTU=115 (EU868,SF9)</b>		
<b>SCHC Header (Rule ID=00010100b, W=00b, FCN=111110b)   SCHC Fragment Payload</b>	143E090E0000000000001180203090C07 E2020C050000000080000110060001 86A00203000006000188400203000006 000189E0020300000600018B80020300 000600018D20020300000600018EC002 03000006000190600203000006000192 000203000006000193A0020300000600 01954002	116
<b>SCHC Fragment 2 (Uplink), Current LoRaWAN MTU=115 (EU868,SF9)</b>		
<b>SCHC Header (Rule ID=00010100b, W=00b, FCN=1111101b)   SCHC Fragment Payload</b>	143D03000006000196E0020300000600 019880020300000600019A2002030000 0600019BC0020300000600019D600203 00000600019F0002030000060001A0A0 02030000060001A24002030000060001 A3E002030000060001A5800203000006 0001A72002030000060001A8C0020300 00060001	116
<b>SCHC Fragment 3 (Uplink), Current LoRaWAN MTU=115 (EU868,SF9)</b>		
<b>SCHC Header (Rule ID=00010100b, W=00b, FCN=1111111b)   SCHC Fragment Payload</b>	143FAA6002030000060001AC00	13
<b>SCHC ACK (Downlink)</b>		
<b>SCHC Header (Rule ID=00010100b, W=00b, C=1)   Padding 5 bits</b>	1440	2
<b>LoRaWAN</b>		
<b>LoRaWAN SCHC Fragment 1 (Uplink)</b>		
<b>LoRaWAN Header (Mtype=010b,RFU=000b, Major=00b)</b>	40	1

Message Elements	Contents	LEN (Bytes)
<b>LoRaWAN FHDR (DevAddr=260116CDh,FCtrl=1000000b,FCnt=0,FOpts =)</b>	CD160126800000	7
<b>LoRaWAN Fport (F/R RuleID)</b>	14	1
<b>LoRaWAN Payload (Encrypted SCHC=3E090E000000000001180203090C07E2020C050 0000000800000110006000186A0020300006000188400 203000006000189E002030000600018B80020300006 00018D2002030000600018EC0020300006000190600 20300000600019200020300006000193A00203000060 001954002) AppSKey = 0x88F282F3F9E901CDC37C3311F01050B2</b>	2748ED7C9F0CE0BA6324C14D11ED8042 0B54CA32891E6816E3B30F7285687309 478E4CB9C04C39AF87B89FA97336D591 9FF7F400D69ADE735BF0B498122538D4 1BDBA07398783B24E04C9CDEC942002A 920B564F33ABE64B9C0322A7E0C8D073 DB0240F498F25E06354C2CC836F0E677 D18E3E	115
<b>MIC NwkSKey = 0x8A69FAD35B3CF856EC86EFE77B3F21C4</b>	673C84CA	4
<b>LoRaWAN (encoded)</b>	40CD160126800000142748ED7C9F0CE0 BA6324C14D11ED80420B54CA32891E68 16E3B30F7285687309478E4CB9C04C39 AF87B89FA97336D5919FF7F400D69ADE 735BF0B498122538D41BDBA07398783B 24E04C9CDEC942002A920B564F33ABE6 4B9C0322A7E0C8D073DB0240F498F25E 06354C2CC836F0E677D18E3E673C84CA	128
<b>LoRaWAN SCHC Fragment 2 (Uplink)</b>		
<b>LoRaWAN Header (Mtype=010b,RFU=000b,Major=00b)</b>	40	1
<b>LoRaWAN FHDR (DevAddr=260116CDh,FCtrl=1000000b,FCnt=1,FOpts =)</b>	CD160126800100	7
<b>LoRaWAN Fport (F/R RuleID)</b>	14	1
<b>LoRaWAN Payload (Encrypted SCHC=3D0300006000196E002030000600019880020 300000600019A2002030000600019BC0020300000600 019D6002030000600019F000203000060001A0A0020 30000060001A2400203000060001A3E0020300000600 01A5800203000060001A7200203000060001A8C0020 30000060001) AppSKey = 0x88F282F3F9E901CDC37C3311F01050B2</b>	A6F8FD9D25D59ED3308393EC55E6BF94 8E013E427A91FCC607CA3AF68ABE23C7 416E15827CE2513FB9A235A9561B17D3 80F2543A560BAF644FBACA1B0E5D5034 8D19BCB83C1F80277FABEAC9583AFF05 9C6E435C1EBAF08FD670F4EA027E24F7 E9FE9B025F059A9D70F69D57C1FE5479 1DFC31	115
<b>MIC NwkSKey = 0x8A69FAD35B3CF856EC86EFE77B3F21C4</b>	28DC4820	4
<b>LoRaWAN (encoded)</b>	40CD16012680010014A6F8FD9D25D59E D3308393EC55E6BF948E013E427A91FC C607CA3AF68ABE23C7416E15827CE251 3FB9A235A9561B17D380F2543A560BAF 644FBACA1B0E5D50348D19BCB83C1F80 277FABEAC9583AFF059C6E435C1EBAF0 8FD670F4EA027E24F7E9FE9B025F059A 9D70F69D57C1FE54791DFC3128DC4820	128
<b>LoRaWAN SCHC Fragment 3 (Uplink)</b>		
<b>LoRaWAN Header (Mtype=010b,RFU=000b,Major=00)</b>	40	1
<b>LoRaWAN FHDR (DevAddr=260116CDh,FCtrl=1000000b,FCnt=2,FOpts =)</b>	CD160126800200	7
<b>LoRaWAN Fport (F/R RuleID)</b>	14	1
<b>LoRaWAN Payload (Encrypted SCHC=3FAA600203000060001AC00) AppSKey = 0x88F282F3F9E901CDC37C3311F01050B2</b>	DB5BD320BDFB419791EF6859	12
<b>MIC NwkSKey = 0x8A69FAD35B3CF856EC86EFE77B3F21C4</b>	363E5E39	4
<b>LoRaWAN (encoded)</b>	40CD16012680020014DB5BD320BDFB41 9791EF6859363E5E39	25
<b>LoRaWAN SCHC ACK (Downlink)</b>		
<b>LoRaWAN Header (Mtype=011b,RFU=000b,Major=00b)</b>	60	1

Message Elements	Contents	LEN (Bytes)
LoRaWAN FHDR (DevAddr=260116CDh,FCtrl=10000000b,FCnt=1,FOpts =)	CD160126800100	7
LoRaWAN Fport (= RuleID)	02	1
LoRaWAN Payload (Encrypted SCHC=40h) AppSKey = 0x88F282F3F9E901CDC37C3311F01050B2	AF	1
MIC (little endian) NwkSKey = 0xA69FAD35B3CF856EC86EFE77B3F21C4	E250164E	4
LoRaWAN (encoded)	60CD16012680010002AFE250164E	14

### 15.3 Example of DLMS/COSEM ACCES service transported through LPWAN with SCHC Fragments

Table 157 – ACCESS service

Message Elements	Contents	LEN (Bytes)
access-request	D9	1
long-invoke-id-and-priority	4000000	4
date-time	00	1
access-request-body		0
access-request-specification		0
SEQUENCE OF CHOICE	04	1
access-request-get	01	1
cosem-attribute-descriptor		0
class-id	0001	2
instance-id	0000600100FF	6
attr-id	02	1
access-request-get	01	1
cosem-attribute-descriptor		0
class-id	0008	2
instance-id	0000010000FF	6
attr-id	02	1
access-request-set	02	1
cosem-attribute-descriptor		0
class-id	0014	2
instance-id	00000D0000FF	6
attr-id	07	1
access-request-set	02	1
cosem-attribute-descriptor		0
class-id	0014	2
instance-id	00000D0000FF	6
attr-id	08	1
access-request-list-of-data		0
SEQUENCE OF Data	04	1
null-data	00	1
null-data	00	1
array	01040203090100090cff01ffffffff f00000000000901000203090101090c ff04ffffffffffff0000000000901010 203090102090cff07ffffffff0000 000000901020203090103090cff0af fffffffff00000000000090103	90
array	0104020809010011ff11ff11ff11ff1 1ff11ff11ff02080901011102110111 011101110111011101020809010211f f11ff11ff11ff11ff11ff11ff020809 0103110111021102110211021102110 2	78



Message Elements	Contents	LEN (Bytes)
SCHC Header (Rule ID=00010101b, W=0b, C=1)   Padding 6 bits	1540	2
SCHC Fragment 2 (Uplink), Current LoRaWAN MTU=115 (EU868, SF9)		
SCHC Header (Rule ID=00010101b, W=0b, FCN=111111b)   SCHC Fragment Payload	15C0240408080C24040C2433FC2BFFFFF FFFFFFC000000000024040C0410082024040 047FC47FC47FC47FC47FC47FC08202 40404440844044404440444044404440444040 82024040847FC47FC47FC47FC47FC4 7FC082024040C440444084408440844084 40844080	106
SCHC ACK (Downlink)		
SCHC Header (Rule ID=00010101b, W=1b, C=1)   Padding 6 bits	15C0	2
LoRaWAN		
LoRaWAN SCHC Fragment 1 (Downlink)		
LoRaWAN Header (Mtype=011b, RFU=000b, Major=00b)	60	1
LoRaWAN FHDR (DevAddr=260116CDh, FCtrl=10000000b, FCnt=0, FOpts=)	CD160126800000	7
LoRaWAN Fport (= RuleID)	15	1
LoRaWAN Payload (Encrypted SCHC=1540) AppSKey = 0x88F282F3F9E901CDC37C3311F01050B2	105ED791164FA6239997B99326A65108FC BA1AD4E90DFCDBB9A1A3C6D45AFAC30FBC CCC6695846DEEA9F3F974BC7C4A4D9683E C7FAA163313C437CA02BA4EC71E1C2D692 9DACE5D671E80B2C1F60161CF711EFBC9C 3FA4C8C23837D0AE6F5135A4B7BA88A9AD 252B09562ECCE3DF5F34D5DB17	115
MIC NwkSKey = 0xA69FAD35B3CF856EC86EFE77B3F21C4	08642724	4
LoRaWAN (encoded)	60CD16012680000015105ED791164FA623 9997B99326A65108FCBA1AD4E90DFCDBB9 A1A3C6D45AFAC30FBCCCC6695846DEEA9F 3F974BC7C4A4D9683EC7FAA163313C437C A02BA4EC71E1C2D692DACE5D671E80B2C 1F60161CF711EFBC9C3FA4C8C23837D0AE 6F5135A4B7BA88A9AD252B09562ECCE3DF 5F34D5DB1708642724	128
LoRaWAN SCHC ACK 1 (Uplink)		
LoRaWAN Header (Mtype=010b, RFU=000b, Major=00b)	40	1
LoRaWAN FHDR (DevAddr=260116CDh, FCtrl=10000000b, FCnt=0, FOpts=)	CD160126800000	7
LoRaWAN Fport (= RuleID)	15	1
LoRaWAN Payload (Encrypted SCHC=1540) AppSKey = 0x88F282F3F9E901CDC37C3311F01050B2	0C01	2
MIC NwkSKey = 0xA69FAD35B3CF856EC86EFE77B3F21C4	79B7BEE6	4
LoRaWAN (encoded)	40CD160126800000150C0179B7BEE6	15
LoRaWAN SCHC Fragment 2 (Downlink)		
LoRaWAN Header (Mtype=011b, RFU=000b, Major=00b)	60	1
LoRaWAN FHDR (DevAddr=260116CDh, FCtrl=10000000b, FCnt=1, FOpts=)	CD160126800100	7
LoRaWAN Fport (= RuleID)	15	1



Message Elements	Contents	LEN (Bytes)
<b>IPv6 (encoded)</b>	60045604001E1180FE800000000000000074 5500FFE000101FE800000000000007455 00FFE0001000FDB0FDB001E3A86000100 010001002ADA400000200000409083030 303030303031090C07DC030C07161E0000 FF8880000040100010002000200	99
<b>SCHC Packet</b>		
<b>Rule ID=0x02 CompressionResidue= SCHC Payload   Padding=</b>	02DA400000200000409083030303030 3031090C07DC030C07161E0000FF888000 00040100010002000200	44
<b>SCHC (encoded)</b>	02DA400000200000409083030303030 3031090C07DC030C07161E0000FF888000 00040100010002000200	44
<b>LoRaWAN (Uplink)</b>		
<b>LoRaWAN Header (Mtype=010b,RFU=000b,Major=00)</b>	40	1
<b>LoRaWAN FHDR (DevAddr=260116CDh,FCtrl=1000000b,FCnt=0001,FO pts=)</b>	CD160126800100	7
<b>LoRaWAN Fport (= RuleID)</b>	02	1
<b>LoRaWAN Payload (Encrypted SCHC=DA40000020000040908303030303031 090C07DC030C07161E0000FF8880000040100010 002000200) AppSKey = 0x88F282F3F9E901CDC37C3311F01050B2</b>	3EB1B322BCFB419598E6F4696744D32466 8795683976EF6453992E8A7B5DD353FF75 EF6DD4D9798DA09CE5	43
<b>MIC (little endian) NwkSKey = 0x8A69FAD35B3CF856EC86EFE77B3F21C4</b>	4EF31528	4
<b>LoRaWAN (encoded)</b>	40CD160126800200023EB1B322BCFB4195 98E6F4696744D324668795683976EF6453 992E8A7B5DD353FF75EF6DD4D9798DA09C E54EF31528	56

**Annex A  
(normative)**  
**NSA Suite B elliptic curves and domain parameters**

NOTE This information is reproduced from NSA2.

Domain parameters  $D$  for ECC schemes are of the form:  $(q, FR, a, b\{, SEED\}, G, n, h)$ , where  $q$  is the field size;  $FR$  is an indication of the basis used;  $a$  and  $b$  are two field elements that define the equation of the curve;  $SEED$  is an optional bit string that is included if the elliptic curve was randomly generated in a verifiable fashion;  $G$  is a generating point consisting of  $(x_G, y_G)$  of prime order on the curve;  $n$  is the order of the point  $G$ ; and  $h$  is the cofactor (which is equal to the order of the curve divided by  $n$ ).

Suite B requires the use of one of the following two sets of domain parameters, see Table A. 1 and

Table A. 2 :

**Table A. 1 – ECC\_P256\_Domain\_Parameters**

Parameter name	Symbol	Value
Field size	$q$	FFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFF = $(2^{224}(2^{32}-1)+2^{192}+2^{96}-1$
Field representation indicator	$FR$	NULL
Curve parameter	$a$	FFFFFFF 00000001 00000000 00000000 00000000 FFFFFFFF FFFFFFFF FFFFFFFC
Curve parameter	$b$	5AC635D8 AA3A93E7 B3EBBD55 769886BC 651D06B0 CC53B0F6 3BCE3C3E 27D2604B
Seed used to generate parameter $b$ :	$SEED$	C49D3608 86E70493 6A6678E1 139D26B7 819F7E90
x-coordinate of base point $G$	$x_G$	6B17D1F2 E12C4247 F8BCE6E5 63A440F2 77037D81 2DEB33A0 F4A13945 D898C296
y-coordinate base point $G$	$y_G$	4FE342E2 FE1A7F9B 8EE7EB4A 7C0F9E16 2BCE3357 6B315ECE CBB64068 37BF51F5
Order of point $G$	$n$	FFFFFFF 00000000 FFFFFFFF FFFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
Cofactor	$h$	1

**Table A. 2 – ECC\_P384\_Domain\_Parameters**

Parameter name	Symbol	Value
Field size	$q$	FFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFF 00000000 00000000 FFFFFFFF = $2^{384}-2^{128}-2^{96}+2^{32}-1$
Field representation indicator	$FR$	NULL
Curve parameter	$a$	FFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFF 00000000 00000000 FFFFFFFC
Curve parameter	$b$	B3312FA7 E23EE7E4 988E056B E3F82D19 181D9C6E FE814112 0314088F 5013875A C656398D 8A2ED19D 2A85C8ED D3EC2AEF
Seed used to generate parameter $b$ :	$SEED$	A335926A A319A27A 1D00896A 6773A482 7ACDAC73

Parameter name	Symbol	Value
x-coordinate of base point G	$x_G$	AA87CA22 BE8B0537 8EB1C71E F320AD74 6E1D3B62 8BA79B98 59F741E0 82542A38 5502F25D BF55296C 3A545E38 72760AB7
y-coordinate base point G	$y_G$	3617DE4A 96262C6F 5D9E98BF 9292DC29 F8F41DBD 289A147C E9DA3113 B5F0B8C0 0A60B1CE 1D7E819D 7A431D7C 90EA0E5F
Order of point G	$n$	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFF FFFFFFFF C7634D81 F4372DDF 581A0DB2 48B0A77A ECEC196A CCC52973
Cofactor	$h$	1

**Annex B**  
**(informative)**  
**Example of an Root-CA and end-entity Certificate**  
**using P-256 signed with P-256**

### B.1 Fields of public key certificates

Table B. 1 shows the fields of public key Certificates. For details, see 9.2.6.4.2.

**Table B. 1 – Fields of public key Certificates using P-256 signed with P-256**

Field	Value	Comments
<i>begin tbsCertificate</i>		
Version	2	X.509 version 3
Serial Number	INTEGER	Assigned by the CA, up to 20 octets
Signature	1.2.840.10045.4.3.2	ecdsa-with-SHA256 (same as signatureAlgorithm)
Issuer	CN, O, C	Distinguished name (DN) of the certificate issuer
Validity	Not before <date> Not after <date>	Validity of the Certificate. Follows RFC 5280.
Subject	CN, O, C	Distinguished name (DN) of the certificate subject
Subject Public Key Info		
AlgorithmIdentifier		
algorithm	1.2.840.10045.2.1	ECDSA and ECDH Public Key
parameters	1.2.840.10045.3.1.7	NIST P-256 named curve
subjectPublicKey	ECC public key bit string 528 bits	1 <sup>st</sup> byte = 0, 2 <sup>nd</sup> byte = 4 (uncompressed) 256 bit x, 256 bit y coordinates
Unique Identifier		
subjectUniqueId	Bit string	Optional in end device certificates other than server certificates.
Extensions		
<b>BasicConstraints</b>		Only required for CA certificates.
Identifier	2.5.29.19	
Value	Boolean   Integer	CA indication and maximum path length
Critical	TRUE	
<b>Subject Key Identifier</b>		Optional in end entity certificates
Identifier	2.5.29.14	
Value	octet string	Follows RFC 5280, 8 or 20 octets.
Critical	FALSE	
<b>Authority Key Identifier</b>		Mandatory in end entity certificates
Identifier	2.5.29.35	
Value	octet string	Follows RFC 5280, 8 or 20 octets.
Critical	FALSE	
<b>Key Usage</b>		Mandatory in all certificates. Values according to RFC 5280, 4.2.1.3: 0 digitalSignature 4 keyAgreement 5 keyCertSign 6 cRLSign At least one bit must be 1.
Identifier	2.5.29.15	
Value	DER encoded bit string	
Critical	TRUE	

Field	Value	Comments
CertificatePolicies Identifier Value Critical	2.5.29.32 OID FALSE	Optional in end entity certificates
Subject Alt Names Identifier Value Critical	2.5.29.17 To be assigned by the PKI or the CA TRUE when Subject is absent.	Optional in end entity certificates
<i>end tbsCertificate</i>		
signatureAlgorithm	1.2.840.10045.4.3.2	ecdsa-with-SHA256
signatureValue	Bit string	Encoded bit string value of a DER encoded sequence of 2 integers; each a maximum of 33 bytes.

## B.2 Example of a Root-CA Certificate using P-256 signed with P-256

This subclause shows an example Root-CA Certificate used to generate the end entity Certificate shown in B.3.

```

Certificate:
  Data:
    Version: 3 (0x2)
    Serial Number: 156 (0x9c)
    Signature Algorithm: ecdsa-with-SHA256
    Issuer: C=CH, O=DLMS USER ASSOCIATION, CN=Root_CA
    Validity
      Not Before: Jan 1 00:00:00 2010 GMT
      Not After : Dec 31 23:59:59 2049 GMT
    Subject: C=CH, O=DLMS USER ASSOCIATION, CN=Root_CA
    Subject Public Key Info:
      Public Key Algorithm: id-ecPublicKey
      Public-Key: (256 bit)
      pub:
        04:09:5d:5d:4d:08:69:82:87:6a:a6:25:f6:c1:67:
        c2:e3:a1:5f:e0:d0:4c:ac:15:c3:f8:66:75:d0:21:
        0a:96:52:fb:3f:96:1c:b6:f3:99:3e:e1:fe:86:7d:
        33:f3:b4:4b:9c:87:60:da:fb:6c:de:63:36:2c:05:
        2c:34:2f:1b:f7
      ASN1 OID: prime256v1
      NIST CURVE: P-256
    X509v3 extensions:
      X509v3 Basic Constraints: critical
      CA:TRUE, pathlen:1
      X509v3 Subject Key Identifier:
        D3:72:AB:23:C7:E0:A4:2C:49:71:EA:B7:D1:B8:3F:20:AB:52:4F:CC
      X509v3 Key Usage: critical
        Certificate Sign, CRL Sign
    Signature Algorithm: ecdsa-with-SHA256
      30:46:02:21:00:aa:d4:0b:10:8d:98:b7:08:4b:a4:ab:9c:f4:
      95:f2:0f:39:c5:0d:1f:32:88:41:36:a7:1c:99:05:ea:67:3d:
      21:02:21:00:ad:f3:67:b0:66:d3:72:53:67:09:ad:26:3f:b8:
      4c:38:73:ae:d8:a7:fd:55:d8:15:14:d4:11:31:6a:20:ad:40
  
```

### B.3 Example of an end entity digital signature Certificate using P-256 signed with P-256

This subclause shows an example end entity Certificate generated using the Root-CA certificate shown in B.2.

Certificate:

  Data:

    Version: 3 (0x2)

    Serial Number: 170 (0xaa)

    Signature Algorithm: ecdsa-with-SHA256

    Issuer: C=CH, O=DLMS USER ASSOCIATION, CN=Root\_CA

    Validity

      Not Before: Jan 1 00:00:00 2010 GMT

      Not After : Dec 31 23:59:59 2049 GMT

    Subject: C=CH, O=DLMS USER ASSOCIATION, CN=4354543030303030

    Subject Public Key Info:

      Public Key Algorithm: id-ecPublicKey

      Public-Key: (256 bit)

      pub:

        04:f4:a3:26:87:23:43:25:7f:57:00:b3:34:88:4b:

        25:97:80:9c:5e:da:ef:0e:eb:b8:9a:1d:c7:5c:6d:

        75:84:0f:4a:12:8b:15:21:5d:b9:3c:51:8c:dd:fe:

        af:25:b2:9a:12:6d:00:e2:d1:98:66:e7:1b:aa:3e:

        51:1b:82:86:81

      ASN1 OID: prime256v1

      NIST CURVE: P-256

    X509v3 extensions:

      X509v3 Authority Key Identifier:

keyid:D3:72:AB:23:C7:E0:A4:2C:49:71:EA:B7:D1:B8:3F:20:AB:52:4F:CC

      X509v3 Key Usage: critical

      Digital Signature

    Signature Algorithm: ecdsa-with-SHA256

      30:46:02:21:00:96:fe:e8:56:dd:53:68:d9:c7:70:c0:da:d5:

      2f:4f:a4:97:12:9a:b7:2c:52:3a:35:66:0e:b6:91:e5:ce:97:

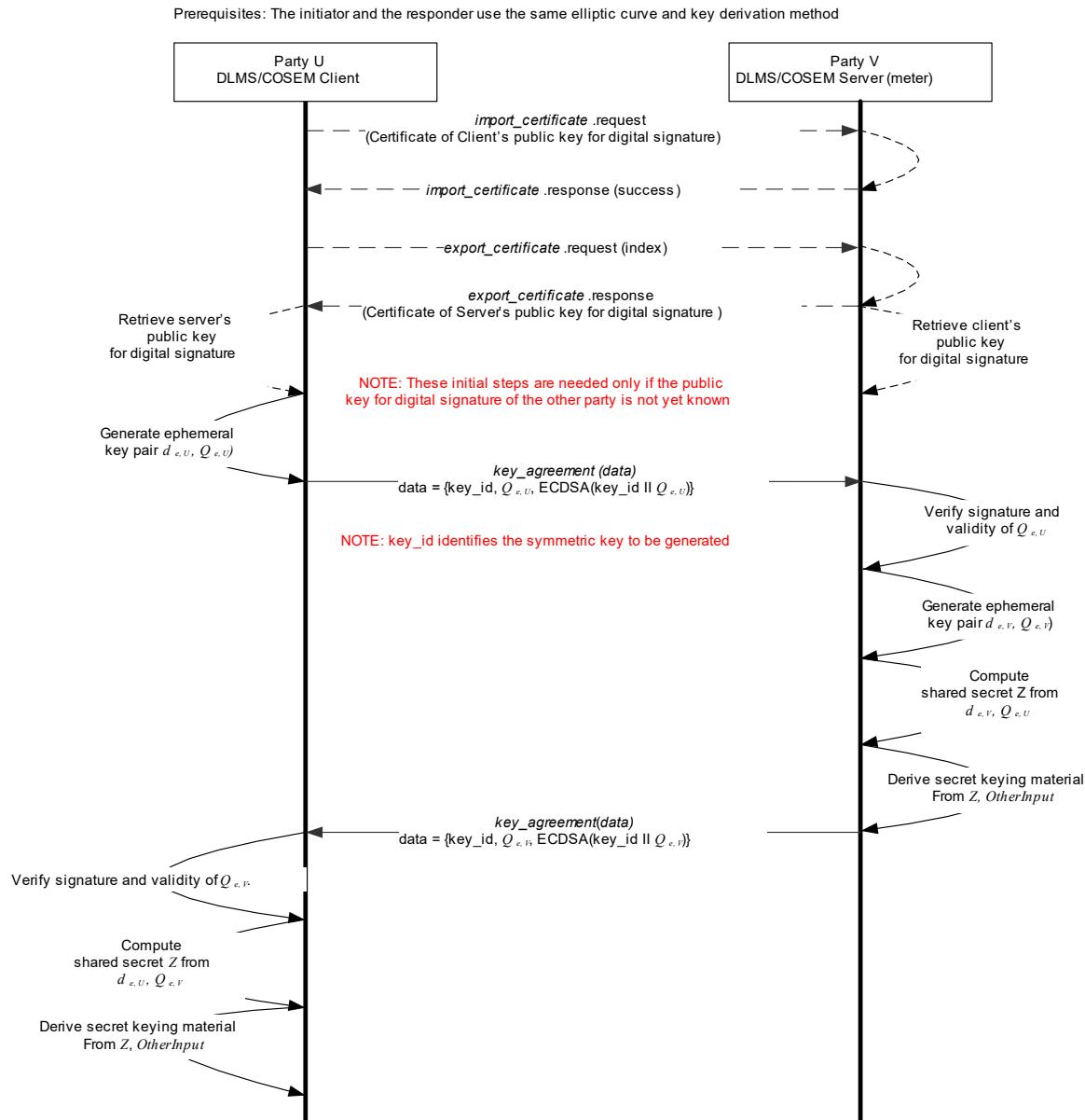
      89:02:21:00:8d:f5:9e:87:58:21:24:7b:ca:6f:18:a2:b3:34:

      aa:39:7b:b8:c0:c4:81:25:7a:f6:17:f5:bb:04:a3:8d:75:a9

**Annex C**  
**(normative)**  
**Use of key agreement schemes in DLMS/COSEM**

### C.1 Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme

Figure C. 1 shows how the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme – specified in 9.2.3.4.6.2 – is used in DLMS/COSEM, by invoking the appropriate methods of the “Security setup” IC. See also 9.2.5.5.



**Figure C. 1 – MSC for key agreement using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme**

The steps are the following (for details, please refer to NIST SP 800-56A Rev. 2: 2013, 6.1.2.2 and NSA2, 3.1):

- Step 1 (optional): the client sends to the server the certificate of its public key for digital signature by invoking the `import_certificate` method;

- Step 2 (optional): the client retrieves from the server the certificate of the public key for digital signature by invoking the *export\_certificate* method;
- Step 3: The client generates an ephemeral key pair ( $d_{e,u}, Q_{e,u}$ ). It signs ( $\text{key\_id}, Q_{e,u}$ ) with its private digital signature key and sends it to the server by invoking the *key\_agreement* method;
- Step 4: The server verifies the signature and the validity of  $Q_{e,u}$ . It computes shared secret  $Z$  from  $(d_{e,v}, Q_{e,u})$  and derives the secret key from  $Z$  and *OtherInput*;
- Step 5: If the key has been successfully derived the server generates then an ephemeral key pair ( $d_{e,v}, Q_{e,v}$ ). It signs ( $\text{key\_id}, Q_{e,v}$ ) and sends it to the client in the response to the invocation of the *key\_agreement* method;
- Step 5: The client computes shared secret  $Z$  from  $(d_{e,u}, Q_{e,v})$  and derives the secret key from  $Z$  and *OtherInput*.

Table C. 1 provides a test vector.

**Table C. 1 – Test vector for key agreement using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme**

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
Security Suite		ECDH-ECDSA-AES-GCM-128-SHA-256		
Curve		P-256		
Domain Parameters	D	See Annex A.		
System Title Client	Sys-TC	4D4D4D0000BC614E	8	64
System Title Server	Sys-TS	4D4D4D0000000001	8	64
Private Signing Key Client	Pri-KC	418073C239FA6125011DE4D6CD2E645780289F761BB21 BFB0835CB5585E8B373	32	256
Private Signing Key Server	Pri-KS	AE55414FFE079F9FC95649536BD1C2B5653D200813727 E07D501A8B550C69207	32	256
Public Signing Key Client	Pub-KC	BAAFFDE06A8CB1C9DAE8D94023C601DBBB249254BA22E DD827E820BCA2BCC64362FBB83D86A82B87BB8B7161D2 AAB5521911A946B97A284A90F7785CD9047D25	64	512
Public Signing Key Server	Pub-KS	933ACF15B03A9248E029B2787FB52A0AECAF635F07C42 A0019FB3197E38F8F549A125EA36781B0CA96BE89A0E1 FE2CF9B7361ED48B3C5E24592B9C0F4EDD31D1	64	512
Ephemeral Public Key Client	Epub-KC	2914D60E10AB705F62ED6CC349D7CB99B9AB3F3978E59 278C7AF595B3AF987941372DAB6D5AF1FA867E134167E 6F23DE664A6693E05F43414611058D1B48F894	64	512
Ephemeral Public Key Server	Epub-KS	95F41066009B185B074F5FFFF736B71C325FCADB2BC0C F1A4F4B17BBE7AB81D62946506BC8169C7B539B39A5D8 463787F449C9BD2583FA67A1075B0DBFC638BA	64	512
Ephemeral Private Key Client	Epri-KC	1BAC19FC1D52A1E5102622EDFA36584C05E12FA8CDEAA 450F2F1E9A7DCCF7628	32	256
Ephemeral Private Key Server	Epri-KS	34A8C23A34DBB519D09B245754C85A6CFE05D14A063EF A5AA41545AA8241EFAE	32	256
Ephemeral Public Key Signature Client	Epub-K-Sig-C	06F0607702AA0E2435A183E2F6B1ECD19629712E389A2 13610C03F77B2590860EA840AF5C3FA1F2BCDF055D474 4E9A01CE9A0E55026BCAA4EEBEB764CED64BB3 // The key_id    Epub-KC are included in signature	64	512

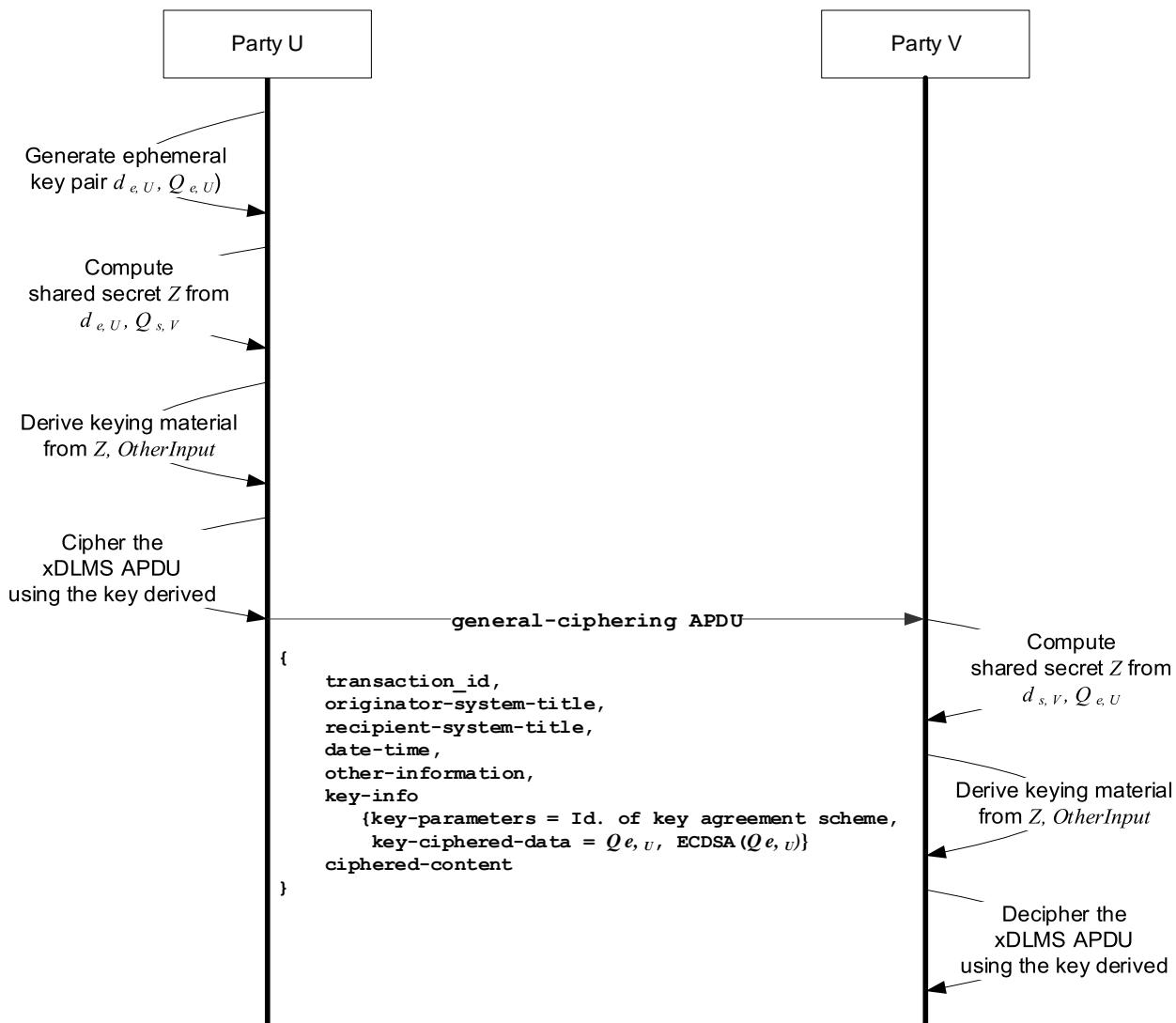
Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
Ephemeral Public Key Signature Server	Epri-K-Sig-S	A92995225CEE004ED4376057EEE9536E97EE6F5BAE43E 59BDBBD515A89FB2CB83F2A270871A31B09338DCF0D17 97466087908BA4A6ED8FD48B9EA067DA67DC4D // The key_id    Epub-KS are included in signature	64	512
key_agreement(data) Client	ACTION-Request	C30140 0040 00002B0000FF 03 // method id 01 // optional flag 0101 // array of 1 0202 // structure of 2 1600 // key_id = 0, global unicast encryption key 098180 2914D60E10AB705F62ED6CC349D7CB99B9AB3F3978E59 278C7AF595B3AF987941372DAB6D5AF1FA867E134167E 6F23DE664A6693E05F43414611058D1B48F894 // ephemeral public key client 64 bytes 06F0607702AA0E2435A183E2F6B1ECD19629712E389A2 13610C03F77B2590860EA840AF5C3FA1F2BCDF055D474 4E9A01CE9A0E55026BCAA4EEBEB764CED64BB3 // ephemeral public key signature client 64 bytes	150	1200
global_key_agreement(data) Server	ACTION-Response	C70140 00 // success 01 // optional Get-Data-result present 00 // data CHOICE 0101 // array of 1 0202 // structure of 2 1600 // key id = 0 098180 // octet string 128 bytes 95F41066009B185B074F5FFFF736B71C325FCADB2BC0C F1A4F4B17BBE7AB81D62946506BC8169C7B539B39A5D8 463787F449C9BD2583FA67A1075B0DBFC638BA // ephemeral public key server 64 bytes A92995225CEE004ED4376057EEE9536E97EE6F5BAE43E 59BDBBD515A89FB2CB83F2A270871A31B09338DCF0D17 97466087908BA4A6ED8FD48B9EA067DA67DC4D // ephemeral public key signature server 64 bytes	143	1144
Shared Secret	Z	C1CF8FE7891AEF3617D7190795E61FE6C24EFC3CCA2E0 8469BAD1A225CE6EA08	32	256
AlgorithmID	AlgID	60857405080300 // AES-GCM-128	7	56
KDF(Z,AlgID,Sys-TC, Sys-TS)	KDF	C5F4512846EDE51CFB8CCF59F08A694E002EDF66B4CB1 739AC26A74E49712F46	32	256
Global Unicast Encryption Key	GUEK	C5F4512846EDE51CFB8CCF59F08A694E	16	128
NOTE The values of the public keys are represented here as FE2OS(xp)   FE2OS(yp).				

## C.2 One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme

Figure C. 2 shows how the One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme, specified in 9.2.3.4.6.3 is used in DLMS/COSEM to protect an xDLMS APDU. See also 9.2.5.5.

Prerequisites:

- Party U and Party V use the same elliptic curve and key derivation method
- Party U has the static public key agreement key  $Q_{s,V}$  of party V



**Figure C. 2 – Ciphered xDLMS APDU protected by an ephemeral key established using the One-pass Diffie-Hellman (1e, 1s, ECC CDH) scheme**

The process is the following:

- Step 1: The originator, taking the role of the party U of the key agreement process generates an ephemeral key pair  $(d_{e,U}, Q_{e,U})$ ;
- Step 2: It computes shared secret  $Z$  from  $d_{e,U}, Q_{s,V}$ ;
- Step 3: It derives the secret key from  $Z$  and *OtherInfo*;
- Step 4: It ciphers the xDLMS APDU as required by the security policy in force and by the access rights, using the key derived;
- Step 5: It sends a general-ciphering APDU to the recipient. The use of the fields of the APDU shall be as follows (see also 9.2.3.4.6.5):
  - transaction-id: as required; not needed for the key derivation process;

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	589/614
-----------------------	------------	-----------------------	---------

- originator-system-title: this is used as the *PartyUInfo* element of *OtherInfo*;
- recipient-system-title: this is used as the *PartyVInfo* element of *OtherInfo*;
- date-time: as required; not needed for the key derivation process;
- other-information: as required; not needed for the key derivation process;
- key-info:
- key-parameters: Identifier of the key agreement scheme: 0x01, see Table 29;
- key-ciphered-data =  $Q_{e, U}$  signed by the digital signature private key of Party U;
- ciphered-content: carries the ciphered xDLMS APDU that is protected using the key.
- Step 6: The recipient, taking the role of party V of the key agreement process computes shared secret Z from  $d_{s, V} Q_{e, U}$ ;
- Step 7: It derives the secret key from Z and *OtherInfo*;
- Step 8: It deciphers the xDLMS APDU using the key derived.

Table C. 2 provides a test vector.

**Table C. 2 – Test vector for key agreement using the One-pass Diffie-Hellman (1e, 1s, ECC CDH) scheme**

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
Security Suite		ECDH-ECDSA-AES-GCM-128-SHA-256		
Curve		P-256		
Domain Parameters	D	See Annex A.		
System Title Client	Sys-TC	4D4D4D0000BC614E	8	64
System Title Server	Sys-TS	4D4D4D0000000001	8	64
Private Signing Key Client	Pri-KC	418073C239FA6125011DE4D6CD2E6 <sup>4</sup> 5780289F761BB21 BFB0835CB5585E8B373	32	256
Private Signing Key Server	Pri-KS	AE55414FFE079F9FC95649536BD1C2B5653D200813727 E07D501A8B550C69207	32	256
Public Signing Key Client	Pub-KC	BAAFFDE06A8CB1C9DAE8D94023C601DBBB249254BA22E DD827E820BCA2BCC64362FBB83D86A82B87BB8B7161D2 AAB5521911A946B97A284A90F7785CD9047D25	64	512
Public Signing Key Server	Pub-KS	933ACF15B03A9248E029B2787FB52A0AECAF635F07C42 A0019FB3197E38F8F549A125EA36781B0CA96BE89A0E1 FE2CF9B7361ED48B3C5E24592B9C0F4EDD31D1	64	512
Private Key Agreement Key Client	Pri-AKC	A51C16FF5C498FCC89323D4A9267CD71BF81FD6F6A891 CD240DA7F3D6F283E65	32	256
Private Key Agreement Key Server	Pri-AKS	AAD3FD0732E991CF52A74C66C1F2827DDC53522A2E0A1 69D7C4FFCC0FB5D6A4D	32	256
Public Key Agreement Key Client	Pub-AKC	07C56DE2DCAF0FD793EF29F019C89B4A0CC1E001CE94F 4FFBE10BC05E7E66F7671A13FBCF9E662B9826FFF6A69 38546D524ED6D3405F020296BDE16B04F7A7C2	64	512
Public Key Agreement Key Server	Pub-AKS	A653565B0E06070BAE9FBE140A5D2156812AEE2DD5250 53E3EFC850BF13BFDFFCB240BC7B77BFF5883344E7275 908D2287BEFA3725017295A096989D2338290B	64	512
Ephemeral Public Key Client	Epub-KC	C323C2BD45711DE4688637D919F92E9DB8FB2DFC213A8 8D21C9DC8DCBA917D8170511DE1BADB360D50058F794B 0960AE11FA28D392CFF907A62D13E3357B1DC0	64	512
Ephemeral Public Key Server	Epub-KS	6439724714B47CD9CB988897D8424AB946DCD083D37A9 54637616011B9C2378773295F0F850D8DAFD1BEBE9FE66 6E53E4F097CD10B38B69622152724A90987444	64	512

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
Ephemeral Private Key Client	Epri-KC	47DAB03842E5B6E74828EF4F449B378D7DD1A5DAE1FFCA5AE0B0BE0AD18EC57E	32	256
Ephemeral Private Key Server	Epri-KS	819B1BEACC955E29139E368BF4119C126FF799EE16BCBA3F45C1EF16749BCB95	32	256
Ephemeral Public Key Signature Client	Epub-K-Sig-C	B51BE089D0B682863B2217201E73A1A9031968A9B4121DCBC3281A69739AF87429F5B3AC5471E7B6A04A2C0F2F8A25FD772A317DF97FC5463FEAC248EB8AB8BE // Epub-KC is included in signature	64	512
Ephemeral Public Key Signature Server	Epub-K-Sig-S	E1FF47974A1F6931A6502F58147463F0E8CC517D47F55B0AC56DD8AC5C9D0E481934F2D90F9893016BD82B6E3FFE21FF1588F3278B4E9D98EB4FB62ADD64B380 // Epub-KS are included in signature	64	512
general-ciphering (Access-Request)	GC-C	DD 080102030405060708 // transaction-id 084D4D4D0000BC614E // originator-system-title 084D4D4D0000000001 // recipient-system-title 00 // date-time not present 00 // other-information not present 01 // optional flag 02 // agreed-key CHOICE 0101 // key-parameters 8180C323C2BD45711DE4688637D919F92E9DB8FB2DFC213A88D21C9DC8DCBA917D8170511DE1BADB360D50058F794B0960AE11FA28D392CF907A62D13E3357B1DC0B51BE089D0B682863B2217201E73A1A9031968A9B4121DCBC3281A69739AF87429F5B3AC5471E7B6A04A2C0F2F8A25FD772A317DF97FC5463FEAC248EB8AB8BE // key-ciphered-data 81EB3100000000F435069679270C5BF4425EE5777402A6C8D51C620EED52DBB188378B836E2857D5C053E6DDF27FA87409AEF502CD9618AE47017C010224FD109CC0BEB21E742D44AB40CD11908743EC90EC8C40E221D517F72228E1A26E827F43DC18ED27B5F458D66508B05A2A4CC6FED178C881AFC3BC67064689BE8BB41C80ABB3C114A31F4CB03B8B64C7E0B4CE77B2399C93347858888F92239713B38DF01C4858245827A92EF334172EA636B31CBBDF2A96AD5D035F66AA38F1A2D97D4BBA99622E6B5F18789CECB2DFB3937D9F3E17F8B472098E6563238F37528374809836002AEA6E7012D2ADFAA7 // ciphered-content	401	3208
general-ciphering (Access-Response)	GC-S	DD 080123456789012345 // transaction-id 084D4D4D0000000001 // originator-system-title 084D4D4D0000BC614E // recipient-system-title 00 // date-time not present 00 // other-information not present 01 // optional flag 02 // agreed-key CHOICE 0101 // key-parameters 81806439724714B47CD9CB988897D8424AB946DCD083D37A954637616011B9C2378773295F0F850D8DAFD1BBE9FE666E53E4F097CD10B38B69622152724A90987444E1FF47974A1F6931A6502F58147463F0E8CC517D47F55B0AC56DD8AC5C9D0E481934F2D90F9893016BD82B6E3FFE21FF1588F3278B4E9D98EB4FB62ADD64B380 // key-ciphered-data 3D3100000000B3FFCAA594642D8319CEC6B2A233E2BF4621D6991B97E4565B986E8CCBE9A299D8E7869723638FF6BB20E66E175E6F2D762CFD26B3D58733 // ciphered-content	226	1808
Shared Secret GC-C	Z-GC-C	0D4385BA0DD756CBCAB9887EB538396EE8F090A14C1079B4359F115B977F4615	32	256

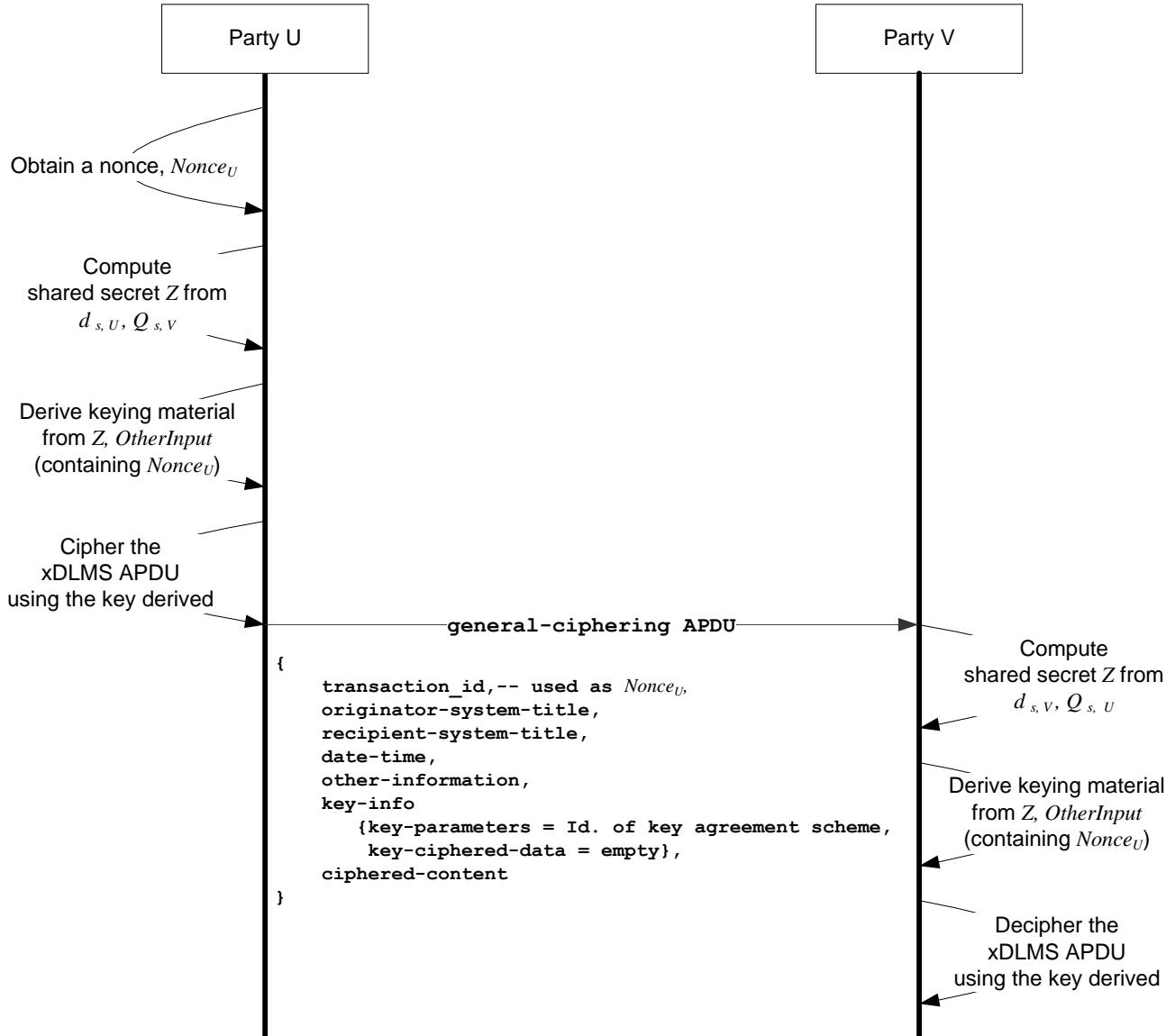
Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
AlgorithmID GC-C	AlgID-GC-C	60857405080300 // AES-GCM-128	7	56
KDF(Z,AlgID,Sys-TC, Sys-TS) GC-C	KDF-GC-C	59A71FD81C929A86A99438DA17A66C058C6A93FD3065F 5EE16A05D775927659B	32	256
Encryption Key GC-C	EK-GC-C	59A71FD81C929A86A99438DA17A66C05	16	128
Shared Secret GC-S	Z-GC-S	2B4302DC49790E2E78D990CFB52ED6E2F273DECE441A2 D95E4301B93812A9FAC	32	256
AlgorithmID GC-S	AlgID-GC-S	60857405080300	7	56
KDF(Z,AlgID,Sys-TS, Sys-TC) GC-S	KDF-GC-S	F0184BDA9466BFA4601A64A7EF46504AB1A40C4851A0A 644503599DF298B2E14	32	256
Encryption Key GC-S	EK-GC-S	F0184BDA9466BFA4601A64A7EF46504A	16	128
NOTE The values of the public keys are represented here as FE2OS(xp)   FE2OS(yp).				

### C.3 Static Unified Model C(0e, 2s, ECC CDH) scheme

Figure C. 3 shows how Static Unified Model C(0e, 2s, ECC CDH) schemes specified in 9.2.3.4.6.4 is used in DLMS/COSEM to protect an xDLMS APDU. See also 9.2.5.5.

Prerequisites:

- Party U and party V use the same elliptic curve and key derivation method.
- Party U and party V have the public key agreement key of the other party



**Figure C. 3 – Ciphered xDLMS APDU protected by an ephemeral key established using the Static Unified Model C(0e, 2s, ECC CDH) scheme**

The process is the following:

- Step 1: The originator, taking the role of the Party U of the key agreement process obtains a nonce,  $\text{Nonce}_U$ ;
- Step 2: It computes shared secret Z from  $d_{s, U}, Q_{s, V}$ ;

NOTE See also Note to Table 24.

- Step 3: It derives the secret key from Z, and *OtherInput* that contains  $\text{Nonce}_U$ ;
- Step 4: It ciphers the xDLMS APDU APDU as required by the security policy in force and by the access rights, using the key derived;
- Step 5: It sends a general-ciphering APDU to the recipient. The use of the fields of the APDU shall be as follows (see also 9.2.3.4.6.5):
  - transaction-id: this field is used as  $\text{Nonce}_U$ ;
  - originator-system-title: this is used as the *PartyUInfo* element of *OtherInfo*;
  - recipient-system-title: this is used as the *PartyVInfo* element of *OtherInfo*;
  - date-time: as required; not needed for the key derivation process;
  - key-info:
  - key-parameters: Identifier of the key agreement scheme: 0x02, see Table 29.
  - key-ciphered-data = empty;
  - ciphered-content carries the ciphered xDLMS APDU that is protected using the key.
- Step 6: The recipient, taking the role of party V of the key agreement process computes shared secret Z from  $d_{s, V}, Q_{s, U}$ ;
- Step 7: It derives the secret key from Z and *OtherInput* that contains  $\text{Nonce}_U$ ;
- Step 8: It deciphers the xDLMS APDU using the key derived.

Table C. 3 provides a test vector.

**Table C. 3 – Test vector for key agreement using the Static-Unified Model (0e, 2s, ECC CDH) scheme**

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
Security Suite		ECDH-ECDSA-AES-GCM-128-SHA-256		
Curve		P-256		
Domain Parameters	D	See Table A.1.		
System Title Client	Sys-TC	4D4D4D0000BC614E	8	64
System Title Server	Sys-TS	4D4D4D0000000001	8	64
Private Key Agreement Key Client	Pri-AKC	A51C16FF5C498FCC89323D4A9267CD71BF81FD6F6A891 CD240DA7F3D6F283E65	32	256
Private Key Agreement Key Server	Pri-AKS	AAD3FD0732E991CF52A74C66C1F2827DDC53522A2E0A1 69D7C4FFCC0FB5D6A4D	32	256
Public Key Agreement Key Client	Pub-AKC	07C56DE2DCAF0FD793EF29F019C89B4A0CC1E001CE94F 4FFBE10BC05E7E66F7671A13FBCF9E662B9826FFF6A69 38546D524ED6D3405F020296BDE16B04F7A7C2	64	512
Public Key Agreement Key Server	Pub-AKS	A653565B0E06070BAE9FBE140A5D2156812AEE2DD5250 53E3EFC850BF13BFDFFCB240BC7B77BFF5883344E7275 908D2287BEFA3725017295A096989D2338290B	64	512
Nonce Client	Nonce-C	080102030405060708 // Nonce-C is transaction-id with length	9	72

Security material	Symbol	Contents	LEN(X) Bytes	LEN(X) Bits
Nonce Server	Nonce-S	080123456789012345 // Nonce-S is transaction-id with length	9	72
general-ciphering (Access-Request)	GC-C	DD 080102030405060708 // transaction-id 084D4D4D0000BC614E // originator-system-title 084D4D4D0000000001 // recipient-system-title 00 // date-time not present 00 // other-information not present 01 // optional flag 02 // agreed-key CHOICE 0102 // key-parameters 00 // key-ciphered-data not present 81EB3100000000607581D83815281B561904E6A72B83B F3FB0B1F2A0A23A82A804B39911F6CB1B4EF9B6F76C63 38ED058014FFBF4A13A162E0EED11EEB8F597757A1821 5F28E1D3FC2D44586C4B8AFE4500B535B579506CDB925 CBEFF7F1CF6BF96C583B9CF588FE8F6B01A574824D7CE C597F057FFA8700AF12AD63A7FA72040439F4392C089B 265F9EB1AC308239906DC04E1A8712ABE383CF9234984 2EBA166EF2E9EB2F53D0DBA025D79875463398BBC3007 BC4A6FC12780C21EE1ABF080BF0FA926242834C0AC30D 55A1EF8856E7DED48621F17FDF4D5B54EDDADD8741192 EB38DA9A2AF773B1E16322DB // ciphered-content	272	2176
general-ciphering(Access-Response)	GC-S	DD 080123456789012345 // transaction-id 084D4D4D0000000001 // originator-system-title 084D4D4D0000BC614E // recipient-system-title 00 // date-time not present 00 // other-information not present 01 // optional flag 02 // agreed-key CHOICE 0102 // key-parameters 00 // key-ciphered-data not present 3D3100000003CF971CD09E0B1C5B89E7BB52C1B39923 D14C4A160D7DDB3F2DE8AD255C625F407F04031CD95F4 F261E23E82E68CC25828C5A901D2681D4D // ciphered-content	97	776
Shared Secret GC-C	Z-GC-C	B1455B2AD5F68BCFFE6AD5412BA89548ACA7E0CBF4B1560D6 A57496F15E931AD	32	256
AlgorithmID GC-C	AlgID-GC-C	60857405080300 // AES-GCM-128	7	56
KDF(Z,AlgID,Sys-TC, Nonce-C, Sys-TS) GC-C	KDF-GC-C	56C46B57DF675515C31025455822514AFA2CDEB3E0BF1 CADA84576159E84DE7E	32	256
Encryption Key GC-C	EK-GC-C	56C46B57DF675515C31025455822514A	16	128
Shared Secret GC-S	Z-GC-S	B1455B2AD5F68BCFFE6AD5412BA89548ACA7E0CBF4B1560D6 A57496F15E931AD	32	256
AlgorithmID GC-S	AlgID-GC-S	60857405080300	7	56
KDF(Z,AlgID,Sys-TS, Nonce-S, Sys-TC) GC-S	KDF-GC-S	FC1314F7DE033B7DD19C80DBCF9FF2C5286DC8F76E878 77BD90B9B7F00CD5613	32	256
Encryption Key GC-S	EK-GC-S	FC1314F7DE033B7DD19C80DBCF9FF2C5	16	128
NOTE The values of the public keys are represented here as FE2OS(xp) II FE2OS(yp).				

**Annex D**  
**(informative)**  
**Exchanging protected xDLMS APDUs between TP and server**

#### D.1 General

This use case shows exchanging protected xDLMS APDUs between a third party (TP) and a server via a client.

#### D.2 Example 1: Protection is the same in the two directions

In the first example, the security policy of the server requires that the request is digitally signed and authenticated and the response is also digitally signed and authenticated.

In the .request, the digital signature is applied by the TP and the authentication is applied by the client:

- the TP sends the .request to the client in a general-signing APDU for the server: the recipient-system-title is that of the server;
- the client verifies the digital signature and if correct, encapsulates the general-signing APDU in a general-ciphering APDU.

The server checks and removes first the authentication and then the digital signature.

If both are correct, it prepares the .response APDU. The protection is applied to the same parties as in the request:

- the server first encapsulates the .response APDU in a general-signing APDU for the TP: the destination-system-title is that of the TP;
- it encapsulates then this general-signing APDU in a general-ciphering APDU for the client: the destination-system-title is that of the client.

The protection to be applied by each party is subject to project specific companion specifications, but the overall protection shall meet the security policy configured in the server. For example, the server would accept any of the following:

- digital signature applied by the TP and authentication applied by the client;
- authentication applied by the TP and digital signature applied by the client;
- both digital signature and authentication applied by the client;
- both digital signature and authentication applied by the TP.

The process is shown in Figure D. 1.

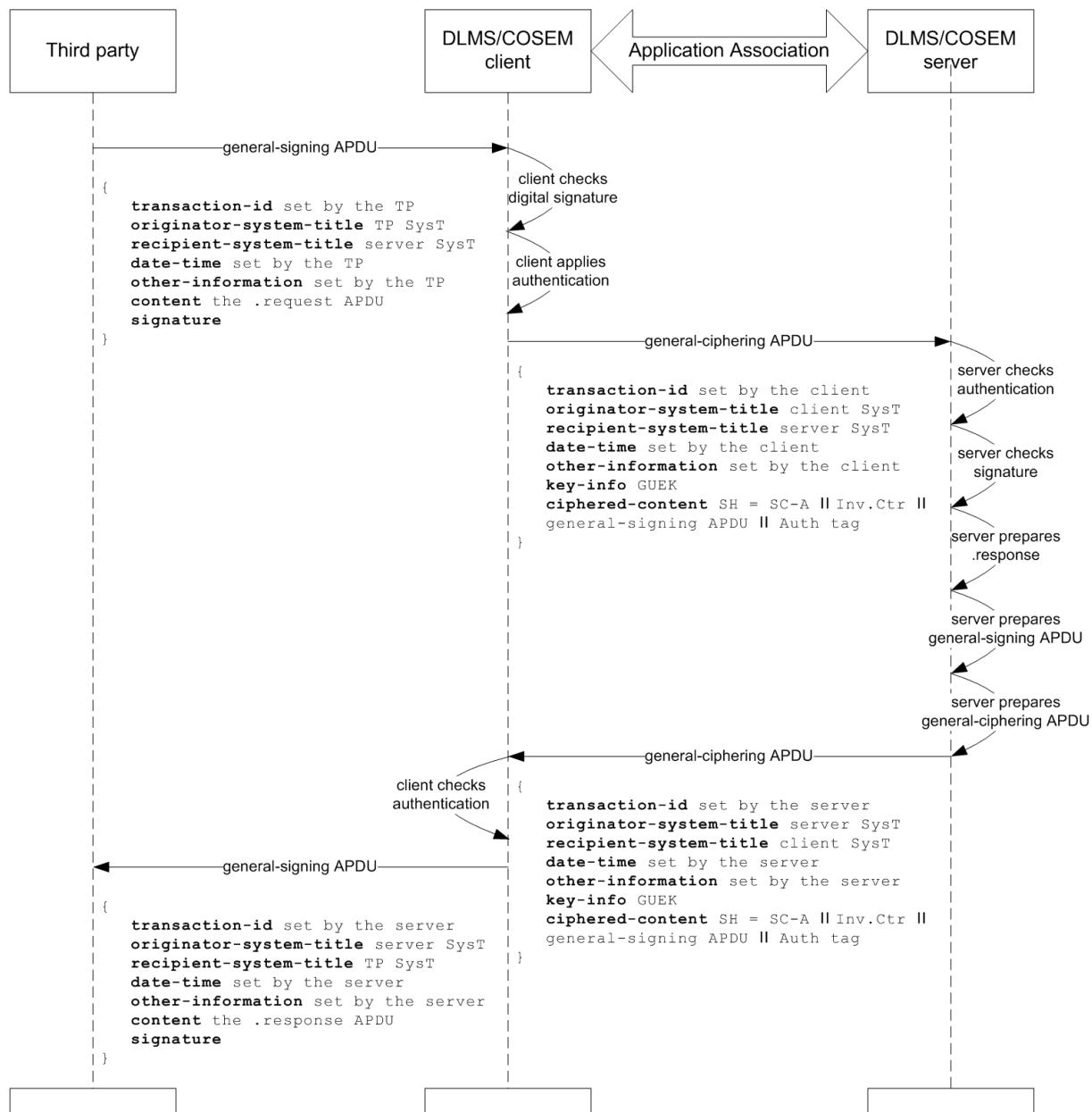


Figure D. 1 – Exchanging protected xDLMS APDUs between TP and server: example 1

### D.3 Example 2: Protection is different in the two directions

In the second example, the security policy of the server requires that the request is digitally signed and authenticated and the response is only authenticated.

In the .request, the digital signature is applied by the TP and the authentication is applied by the client:

- the TP sends the .request to the client in a general-signing APDU for the server: the recipient-system-title is that of the server;
- the client verifies the digital signature and if correct, encapsulates the general-signing APDU in a general-ciphering APDU.

The server checks and removes first the authentication tag and then the digital signature.

If both are correct, it prepares the .response APDU. The protection is applied to the same parties as in the request:

- the server first encapsulates the .response APDU in a general-ciphering APDU for the TP: the destination-system-title is that of the TP, but no protection is applied: in the Security Control Byte, the bits indicating authentication and encryption are set to 0;
- it encapsulates then this general-ciphering APDU in a general-ciphering APDU for the client: the destination-system-title is that of the client.

The process is shown in Figure D. 2.

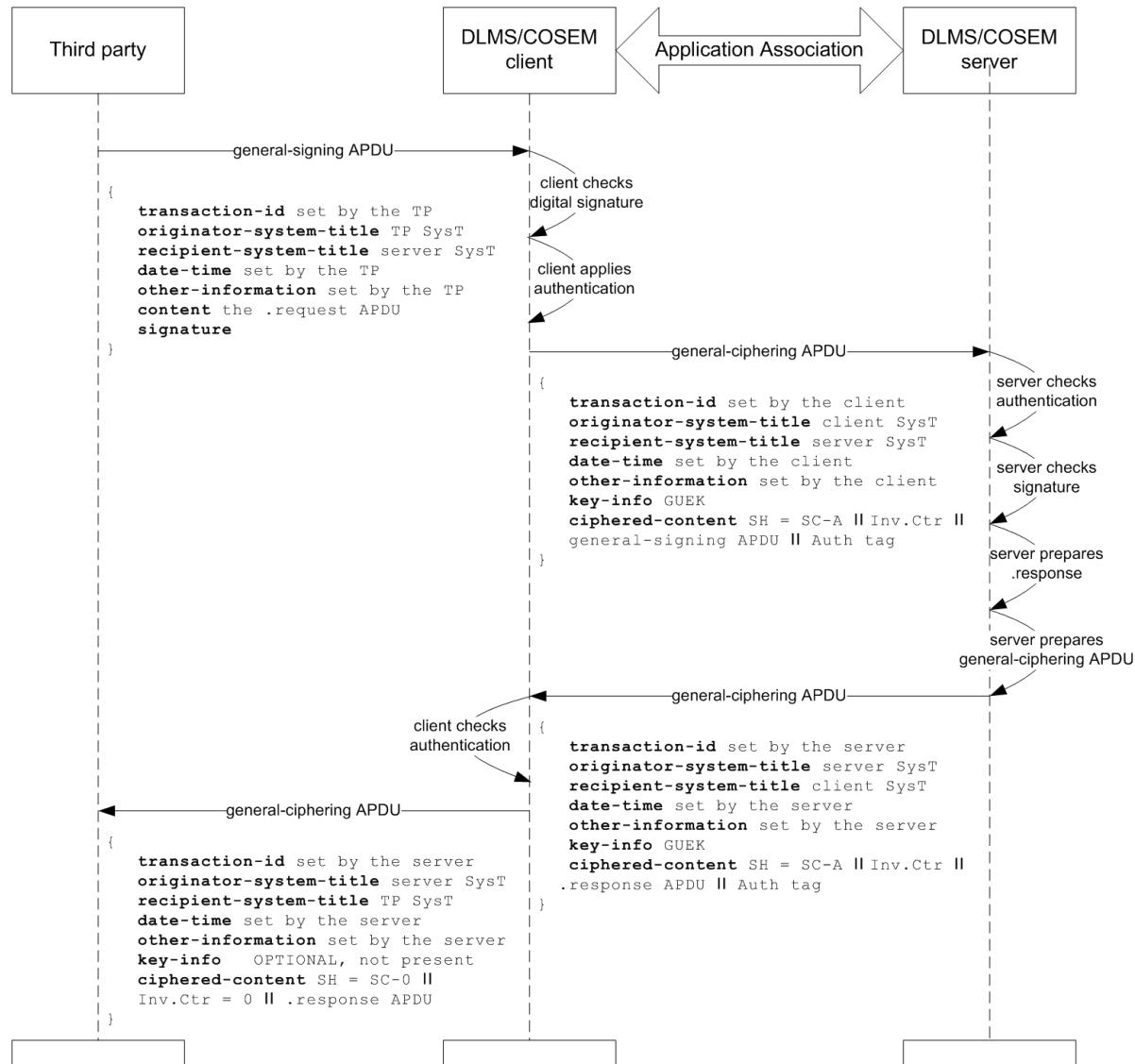


Figure D. 2 – Exchanging protected xDLMS APDUs between TP and server: example 2

## Bibliography

IEC 60050-300, *International Electrotechnical Vocabulary – Revision of Chapter 301, 302, 303 – Electrical measurements and measuring instruments - Chapter 311: General terms relating to measurements Chapter 312: General terms relating to electrical measurements - Chapter 313: Types of electrical measuring instrument - Chapter 314: Specific terms according to the type of instrument*

IEC 60870-5-1:1990, *Telecontrol equipment and systems. Part 5: Transmission protocols – Section One: Transmission frame formats*

IEC 61334-4-512:2001, *Distribution automation using distribution line carrier systems – Part 4-512: Data communication protocols – System management using profile 61334-5-1 – Management Information Base (MIB)*

IEC 62051:1999, *Electricity metering – Glossary of terms*

IEC/TR 62051-1:2004, *Electricity metering – Data exchange for meter reading, tariff and load control – Glossary of Terms - Part 1, Terms related to data exchange with metering equipment using DLMS/COSEM*

IEC TS 62056-1-1, *Electricity metering data exchange – The DLMS/COSEM suite – Part 1-1: Template for DLMS/COSEM communication profile standards*

IEC 62056-3-1:2013, *Electricity metering data exchange - The DLMS/COSEM suite - Part 3-1: Use of local area networks on twisted pair with carrier signalling*

IEC 62056-41:1998, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 41: Data exchange using wide area networks: Public switched telephone network (PSTN) with LINK+ protocol*

IEC 62056-42:2002, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 42: Physical layer services and procedures for connection-oriented asynchronous data exchange*

IEC 62056-46:2007, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 46: Data link layer using HDLC protocol*

IEC 62056-47:2006, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 47: COSEM transport layer for IP networks*

IEC 62056-4-7, *ELECTRICITY METERING DATA EXCHANGE – The DLMS/COSEM suite – Part 4-7: DLMS/COSEM transport layer for IP networks*

This standard cancels and replaces IEC 62056-47:2006.

IEC/TS 62056-51:1998, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 51: Application layer protocols*

IEC/TS 62056-52:1998, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 52: Communication protocols management distribution line message specification (DLMS) server*

IEC 62056-5-3, *Electricity metering data exchange – The DLMS/COSEM suite – Part 5-3: DLMS/COSEM application layer*

IEC 62056-6-1, *Electricity metering data exchange – The DLMS/COSEM suite – Part 6-1: Object Identification System (OBIS)*

IEC 62056-6-2, *Electricity metering data exchange – The DLMS/COSEM suite – Part 6-2: COSEM interface classes*

IEC 62056-7-3, *Electricity metering data exchange – The DLMS/COSEM suite – Part 7-3: Wired and wireless M-Bus communication profiles for local end neighbourhood networks*

IEC 62056-7-6:2013, *ELECTRICITY METERING DATA EXCHANGE – The DLMS/COSEM suite – Part 7-6: The 3-layer, connection-oriented HDLC based communication profile*

IEC 62056-8-3, *ELECTRICITY METERING DATA EXCHANGE – The DLMS/COSEM suite – Part 8-3: PLC S-FSK communication profile for neighbourhood networks*

IEC 62056-8-4, *ELECTRICITY METERING DATA EXCHANGE – THE DLMS/COSEM SUITE – Part 8-4: Narrow-band OFDM PRIME PLC communication profile for neighbourhood networks*

IEC 62056-8-5, *ELECTRICITY METERING DATA EXCHANGE – THE DLMS/COSEM SUITE – Part 8-5: Narrow-band OFDM G3-PLC communication profile for neighbourhood networks*

IEC 62056-9-7, *ELECTRICITY METERING DATA EXCHANGE – The DLMS/COSEM suite – Part 9-7: Communication profile for TCP-UDP/IP networks*

ISO/IEC 9545:1994, *Information technology - Open Systems Interconnection – Application layer structure*

Evaluation of ISO/IEC 9798 Protocols Version 2.0 David Basin and Cas Cremers April 7, 2011

ISO/IEC 9798-1:2010, *Information technology — Security techniques — Entity authentication — Part 1: General*

ISO/IEC 9798-2 Ed. 3:2008, *Information technology — Security techniques — Entity authentication — Part 2: Mechanisms using symmetric encipherment algorithms*

ISO/IEC 9798-3:1998, *Information technology – Security techniques – Entity authentication – Part 3: Mechanisms using digital signature techniques*

ISO/IEC 10731:1994, *Information technology - Open Systems Interconnection - Basic Reference Model - Conventions for the definition of OSI services*

ISO/IEC 15945:2002, *Information technology — Security techniques — Specification of TTP services to support the application of digital signatures*

ISO 2110:1989, *Information technology – Data communication – 25-pole DTE/DCE interface connector and contact number assignments*

ITU-T V.24:1996, *List of definitions for interchange circuits between data terminal equipment (DTE) and data circuit-terminating equipment (DCE)*

ITU-T V.25:1996, *Automatic answering equipment and general procedures for automatic calling equipment on the general switched telephone network*

ITU-T V.25bis:1996, *Synchronous and asynchronous automatic dialling procedures on switched networks*

ITU-T V.28:1993, *Electrical characteristics for unbalanced double-current interchange circuits*

ITU-T X.811:1995, *Information technology - Open Systems Interconnection - Security frameworks for open systems: Authentication framework*

IEEE 802.1 ae:2006, *IEEE Standard for Local and metropolitan area networks Media Access Control (MAC) Security*

IEEE 802.15.4:2006, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications*

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	601/614
-----------------------	------------	-----------------------	---------

CEN/CLC/ETSI TR 50572 *Functional reference architecture for communications in smart metering systems*

ANSI/TIA 4957.200, *Layer 2 Standard Specification for the Smart Utility Network*

ANSI/TIA 4957.210, *Multi-Hop Sublayer Specification-Extension on Field Area Networks*

FIPS PUB 198:2002, *The Keyed-Hash Message Authentication Code (HMAC)*

FIPS PUB 199:2002, *Standards for Security Categorization of Federal Information and Information Systems*

NIST SP 800-47:2002, *Security Guide for Interconnecting Information Technology Systems*

*The Galois/Counter Mode of Operation (GCM)* - David A. McGrew, Cisco Systems, Inc. 170, West Tasman Drive, San Jose, CA 95032, [mcgrew@cisco.com](mailto:mcgrew@cisco.com), John Viega, Secure Software, 4100 Lafayette Center Drive, Suite 100, Chantilly, VA 20151, [viega@securesoftware.com](mailto:viega@securesoftware.com), May 31, 2005

RFC 0791, *Internet Protocol*, 1981, Also: STD0005, Updated by: RFC 1349, Obsoletes: RFC 0760, <http://tools.ietf.org/html/rfc791>

RFC 0792, *Internet Control Message Protocol*, 1981, Also: STD0005, Updated by: RFC 0950, Obsoletes: RFC0777, <http://tools.ietf.org/html/rfc792>

RFC 0793, *Transmission Control Protocol*, 1981, Also: STD0007, Updated by: RFC 3168, <http://tools.ietf.org/html/rfc793>

RFC 0822, *Standard for the format of ARPA Internet Text Messages*, 1982, <http://www.ietf.org/rfc/rfc822>

RFC 0826, *Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*, 1982, Also: STD0037, <http://tools.ietf.org/html/rfc826>

RFC 0894, *Standard for the transmission of IP datagrams over Ethernet networks*, 1984, Also: STD0041, <http://tools.ietf.org/html/rfc894>

RFC 0919, *Broadcasting Internet Datagrams*, 1984, Also: STD0005, <http://tools.ietf.org/html/rfc919>

RFC 0922, *Broadcasting Internet datagrams in the presence of subnets*, 1984, Also: STD0005, <http://tools.ietf.org/html/rfc922>

RFC 0950, *Internet Standard Subnetting Procedure*, 1985, Also: STD0005, Updates: RFC 0792, <http://tools.ietf.org/html/rfc950>

RFC 1042, *Standard for the transmission of IP datagrams over IEEE 802 networks*, 1988, Also: STD0043, Obsoletes: RFC0948, <http://www.ietf.org/rfc/rfc1042.txt>

RFC 1095, *The Common Management Information Services and Protocol over TCP/IP, (CMOT)*, 1989, <http://www.ietf.org/rfc/rfc1095>

RFC 1112, *Host extensions for IP multicasting*, 1989, Also: STD0005, Updated by: RFC 2236, Obsoletes: RFC0988, RFC1054, <https://tools.ietf.org/rfc/rfc1112.txt>

RFC 1321, *The MD5 Message-Digest Algorithm*, 1982, <http://www.ietf.org/rfc/rfc1321.txt>

RFC 1662, *PPP in HDLC-like Framing*, 1984, <http://www.ietf.org/rfc/rfc1662.txt>

RFC 2104, *HMAC: Keyed-Hashing for Message Authentication*, 2004, <http://www.ietf.org/rfc/rfc2104.txt>

RFC 2119, Key words for use in RFCs to Indicate Requirement Levels, 1997, <http://www.ietf.org/rfc/rfc2119.txt>

RFC 2315, PKCS #7, Cryptographic Message Syntax Version 1.5, 1998, <https://www.ietf.org/rfc/rfc2315>

RFC 2560 X.509, Internet Public Key Infrastructure – Online Certificate Status Protocol – OCSP, 1999, <http://www.ietf.org/rfc/rfc2560>

RFC 2822, Internet Message Format, 2001, <http://www.ietf.org/rfc/rfc2822>

RFC 2986, PKCS #10 v1.7: Certification Request Syntax Standard, <http://www.ietf.org/rfc/rfc2986>

RFC 3268, Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS), 2002, <http://tools.ietf.org/html/rfc3268>

RFC 4106, The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP), 2005, <https://www.rfc-editor.org/rfc/rfc4106.txt>

RFC 4210, Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP), 2005, <http://www.ietf.org/rfc/rfc4210.txt>

RFC 4211, Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF), 2005, <http://www.ietf.org/rfc/rfc4211>

RFC 4308, Cryptographic Suites for IPsec, 2005, <https://www.google.hu/#q=RFC%204308>

RFC 4835, Cryptographic Algorithm Implementation Requirements for Encapsulating Security Payload (ESP) and Authentication Header (AH), 2007, <http://tools.ietf.org/html/rfc4335>

RFC 5084, Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS), 2007, <https://www.google.hu/#q=RFC+5084>

RFC 5349, Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT), 2008, <https://tools.ietf.org/html/rfc5349>

RFC 5480, Elliptic Curve Cryptography Subject Public Key Information, 2009, <http://www.ietf.org/rfc/rfc5480.txt>

RFC 5758, Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA, 2010, <http://www.ietf.org/rfc/rfc5758.txt>

RFC 5759, Suite B Certificate and Certificate Revocation List (CRL) Profile, 2010, <http://tools.ietf.org/search/rfc5759>

RFC 6024, Trust Anchor Management Requirements <http://www.ietf.org/rfc/rfc6024.txt>

RFC 6318, Suite B in Secure/Multipurpose Internet Mail Extensions (S/MIME), 2011, <http://tools.ietf.org/html/rfc6318>

RFC 7967 Constrained Application Protocol (CoAP) Option for No Server Response

SEC1:2009, Standards for Efficient Cryptography: Elliptic Curve Cryptography. SECG. Version 2.0

SEC2:2010, Standards for Efficient Cryptography: Recommended Elliptic Curve Domain Parameters, Version 2.0. Certicom Research

UK DECC Smart Metering Implementation Programme – Great Britain Companion Specification (GBCS) V0.7 Rev6

Technische Richtlinie BSI TR-03109-4: Public Key Infrastruktur für Smart Meter Gateways Version 1 – 18.03.2013. Publicly available at:

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	603/614
-----------------------	------------	-----------------------	---------

[https://www.bsi.bund.de/DE/Themen/SmartMeter/TechnRichtlinie/TR\\_node.html](https://www.bsi.bund.de/DE/Themen/SmartMeter/TechnRichtlinie/TR_node.html)

## Index

- 16-bit FCS computation method ..... 188
- 3-layer, connection-oriented, HDLC based communication profile ..... 408
- AA, confirmed ..... 195, 199
- AA, pre-established ..... 195, 199
- AA, unconfirmed ..... 195, 199
- AARE APDU, BER encoding ..... 506, 515
- AARQ and AARE encoding examples ..... 497
- AARQ APDU, BER encoding ..... 503, 513
- A-ASSOCIATE service ..... 195, 264, 308
- Abort sequence ..... 175
- Abstract syntax ..... 446
- Abstract Syntax Notation 1 ..... 384
- Abstract syntax, COSEM APDUs ..... 370
- Access right ..... 18, 50, 51, 204, 207, 247
- ACCESS service ..... 197, 286, 331, 548
- Access\_Request\_Action ..... 290
- Access\_Request\_Body ..... 289
- Access\_Request\_Get ..... 290
- Access\_Request\_List\_Of\_Data ..... 290
- Access\_Request\_Set ..... 290
- Access\_Request\_Specification ..... 289, 290
- Access\_Response\_Body ..... 290
- Access\_Response\_List\_Of\_Data ..... 291
- Access\_Selection\_Parameters ..... 279, 281
- ACSE functional units ..... 307
- ACSE procedures ..... 203
- ACSE protocol version ..... 266, 317
- ACSE requirements ..... 309
- ACSE services and APDUs ..... 307
- ACSE, connection oriented ..... 195
- ACSE\_Requirements ..... 267
- ACTION service ..... 197, 282, 329
- ACTION.confirm ..... 286
- ACTION.indication ..... 286
- ACTION.request ..... 286
- ACTION.response ..... 286
- Action-Request ..... 286
- ACTION-REQUEST-FIRST-BLOCK284, 330, 337, 342
- ACTION-REQUEST-LAST-BLOCK284, 330, 338, 342
- ACTION-REQUEST-NEXT ..... 284, 330, 337
- Action-Request-Next-Pblock ..... 330
- Action-Request-Normal ..... 330
- ACTION-REQUEST-NORMAL284, 330, 337, 342, 345
- ACTION-REQUEST-ONE-BLOCK284, 330, 338, 342
- Action-Request-With-First-Pblock ..... 330
- Action-Request-With-List ..... 330
- ACTION-REQUEST-WITH-LIST284, 330, 338, 342, 345
- ACTION-REQUEST-WITH-LIST-AND-FIRST-BLOCK ..... 284, 330, 338, 342
- Action-Request-With-List-And-With-Frist-Pblock ..... 330
- Action-Request-With-Pblock ..... 330
- Action-Response ..... 286
- ACTION-RESPONSE-LAST-BLOCK284, 330, 338
- ACTION-RESPONSE-NEXT284, 330, 338, 342
- Action-Response-Next-Pblock ..... 330
- Action-Response-Normal ..... 330
- ACTION-RESPONSE-NORMAL284, 330, 338, 342
- ACTION-RESPONSE-ONE-BLOCK ... 284, 330
- ACTION-RESPONSE-ONE-ONE-BLOCK ... 338
- Action-Response-With-List ..... 330
- ACTION-RESPONSE-WITH-LIST284, 330, 338, 342
- Action-Response-With-Pblock ..... 330
- Active HDLC channel state ..... 174
- Active initiator ..... 429, 436
- active OPEN* ..... 98
- Additional Authenticated Data214, 216, 228, 250
- Additional data ..... 214
- Address lengths, inopportune ..... 170
- Addressing ..... 47, 416
- Addressing capability (wPort) ..... 79
- Addressing, HDLC ..... 168
- Addressing, M-Bus ..... 464
- Addressing, S-FSK ..... 446
- Advanced Encryption Standard ..212, 213, 227
- AES key wrap ..... 216
- AES-GCM algorithm ..... 247, 249
- Agreed\_Key\_Options ..... 277
- agreed-key ..... 230
- AL, management services ..... 303
- ALARM state ..... 448
- Alarm\_Descriptor ..... 439
- AlgorithmID ..... 225
- All Physical ..... 447
- All-configured ..... 447
- All-L-SAP ..... 447
- All-station (Broadcast) address ..... 169
- All-station address ..... 170
- Always repeater ..... 439
- Application association ..... 45, 49, 194, 269
- Application association, confirmed50, 313, 419
- Application association, confirmed AA ..... 411
- Application association, establishment ..... 313, 449, 453
- Application association, graceful release ... 318
- Application association, non-graceful release ..... 321
- Application association, pre-established ..... 49, 317
- Application association, release ..... 317, 451
- Application association, unconfirmed50, 317, 411, 419
- Application context ..... 50
- Application context name ... 195, 266, 310, 315

DLMS User Association	2021-12-22	DLMS UA 1000-2 Ed. 11	605/614
-----------------------	------------	-----------------------	---------

Application entity .....	45
Application layer .....	433
Application layer message security .....	207
Application process .....	45, 52, 194, 433
Application process identification .....	93
Application Programming Interface .....	198
Application Service Object .....	194
Application_Addresses parameter .....	293
application-context-name .....	309
A-RELEASE .....	269
A-RELEASE service .....	195, 308, 420
ASN.1 .....	311
Association Control Service Element .....	194, 195
Association LN .....	45
Association SN .....	45
Asymmetric key algorithm .....	210
Attribute .....	17
Attribute_0 referencing .....	198
Authenticated decryption .....	214
Authenticated encryption .....	213, 214
Authentication18, 50, 203, 204, 207, 210, 211, 217, 227, 249	
Authentication functional unit .....	307
Authentication key .....	216, 228, 250, 255
Authentication key, GAK .....	228, 230, 231
Authentication mechanism .....	50, 204
Authentication mechanism name .....	195, 311
Authentication tag .....	214, 215, 216
Authentication, High Level Security206, 267, 316	
Authentication, Low Level Security .....	206, 267
Authentication, Lowest Level Security 206, 267	
Authentication, no security .....	206
Authenticity .....	211
Automatic configuration, Repeater Call service .....	436
A-XDR encoding .....	66
Battery operated devices .....	471
Bit String .....	218
Block cipher .....	212
Block cipher algorithm .....	212
Block cipher key .....	216, 228
Block transfer, general .....	201, 324, 329
Block transfer, service-specific .....	324, 329
Block_Number279, 282, 285, 297, 298, 300, 325, 329	
Block_Number_Access .....	294, 296, 297
Block_Transfer_Streaming .....	348
Block_Transfer_Window .....	348
block-data .....	348
block-number .....	348
block-number-acknowledged .....	348
Broadcasting .....	452, 454
Broker .....	52, 209, 233
Busy .....	186
Called_AE_Invocation_Identifier .....	266
Called_AE_Qualifier .....	266
Called_AP_Invocation .....	266
Called_AP_Title .....	266
Calling authentication value .....	267
CALLING Physical Device .....	170
CALLING Physical Device Address .....	169, 414
calling_AE_invocation_identifier: .....	315
Calling_AE_Qualifier .....	266
Calling_AP_Invocation_Identifier .....	266
calling-AE-qualifier .....	315
calling-authentication-value .....	309
Certificate and certificate extension profile232, 235	
Certificate extension .....	239
Certificate extension, Authority Key Identifier .....	240
Certificate extension, Basic constraints .....	241
Certificate extension, CertificatePolicies .....	240
Certificate extension, cRLDistributionPoints .....	242
Certificate extension, Extended Key Usage .....	242
Certificate extension, IssuerAltName .....	241
Certificate extension, KeyUsage .....	240
Certificate extension, SubjectAltNames .....	240
Certificate extension, SubjectKeyIdentifier .....	240
Certificate Policy .....	234, 235
Certificate removal .....	246
Certificate Signing Request .....	233
Certificate, client .....	245
Certificate, digital signature key .....	235
Certificate, Issuer .....	237
Certificate, Serial number .....	237
Certificate, static key agreement key .....	235
Certificate, Subject .....	237
Certificate, Subject Unique ID .....	239
Certificate, SubjectPublicKeyInfo .....	238
Certificate, Validity period .....	238
Certificate, X.509 v3 .....	235
Certificates, server .....	245
Certification Authority .....	233
Challenge, CtoS .....	206
Challenge, StoC .....	206
change_HLS_secret .....	207
Check Initiator Phase .....	440, 443, 445
CIASE .....	447
CIASE L-SAP .....	447
CIASE protocol .....	48
CI-PDU .....	519
Ciphered APDU .....	18
Ciphered xDLMS APDUs .....	248
Ciphertext .....	211, 215, 227
Clear Alarm, S-FSK .....	529
ClearAlarm service .....	434, 439
Client .....	45, 46, 48, 49, 51, 52, 258
Client Management Process48, 85, 169, 447, 448, 468, 480, 483	
Client side layer management services .....	303
Client SN Mapper ASE .....	300, 302
Client SN_Mapper ASE .....	194, 298
Client system title .....	315
Client user .....	266
Client user identification .....	49, 204, 315
Client/server model .....	46
Client_LLC_Address .....	409

Client_MAC_Address.....	409
Client_Max_Receive_PDU_Size .....	267
client_system_title.....	49
client-max-receive-pdu-size .....	315
Collision.....	414
Command/response frame rejection .....	186
Common Name.....	237
Communication environment.....	407
Communication profile .....	52
Communication profile specific parameters	407
Communication profile structure.....	407
Compact array.....	549
Composable xDLMS messages.....	200
Compression.....	45, 195, 200, 226, 249
Confidentiality .....	210, 211
Configuration Initiation Application Service Element .....	434
Confirmed service .....	196
Confirmed service invocations .....	322
confirmedServiceError .....	298, 300, 323
Conformance block.....	202, 287, 321, 498
Conformance testing .....	58
Connection oriented .....	49
Context negotiation .....	195
Control field .....	167
Control function.....	194, 304
Copyright .....	13
COSEM APDU, abstract syntax.....	370
COSEM application context .....	199, 202
COSEM application context name .....	310, 311
COSEM authentication mechanism .....	50
COSEM authentication mechanism name .....	311, 312
COSEM Cryptographic algorithm ID-s .....	312
COSEM data protection .....	261
COSEM data security .....	210
COSEM Glossary of Terms.....	18
COSEM logical device address .....	409
COSEM object18, 38, 45, 50, 51, 52, 54, 58, 196, 197, 198, 204, 206, 207, 209, 210, 247, 248, 261, 264, 277, 279, 280, 281, 284, 286, 287, 290, 291, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 322, 323, 327, 331, 336, 340, 344	
COSEM object model .....	45
COSEM physical device address.....	409
COSEM_Attribute_Descriptor.....	279, 281, 290
COSEM_Authentication_Mechanism_Name .....	316
COSEM_Class_Id.....	279, 281, 284
COSEM_Method_Descriptor .....	284
COSEM_Method_Id.....	284
COSEM_Object_Attribute_Id.....	279, 281
COSEM_Object_Instance_Id.....	279, 281, 284
COSEM_on_IP .....	53, 415
COSEM-ABORT .....	411
COSEM-ABORT service .....	195, 272
COSEM-OPEN service	195, 264, 313, 411, 420
COSEM-OPEN service invocations, repeated .....	317
COSEM-RELEASE service.	195, 269, 318, 411
Counter mode.....	213
Country.....	237
Critical certificate extension .....	235
Cross-talk.....	440
Cryptographic algorithm .....	210
Cryptographic algorithm IDs.....	312
Cryptographic protection .....	195
Cryptoperiod.....	231
Data .....	261
Data collection system .....	54
Data Communication Equipment .....	59
Data conversions .....	218
Data integrity .....	210, 217
Data length.....	79, 84
Data link connection, disconnecting.....	157
Data link connection, setting up .....	153
Data link layer, HDLC .....	77, 151, 408
Data link layer, management services .....	191
Data link layer, S-FSK .....	432
Data link, disconnection .....	178
Data Terminal Equipment.....	59
Data transfer .....	161
Data transfer services, protocol.....	321
Data_Access_Error.....	297, 300
Data_Access_Result.....	282
Data_Length.....	82
DataBlock_G .....	279, 325, 326
DataBlock_SA .....	282, 285, 328
DataNotification service	51, 196, 197, 292, 333, 470
Date_Time .....	277
date-time.....	254
Decryption.....	211
Dedicated key .....	228, 267
Denial-of-service attack .....	270, 451
Destination address .....	167, 170
Destination wPort .....	84
Deterministic construction .....	215
Device identification number .....	466
Device type .....	466
Device version.....	466
Digital signature .....	207, 217, 219, 227, 258
Digital signature key pair .....	232
Direct local connection.....	74
Direct local data exchange .....	74
Directly trusted key .....	233
DISC command .....	178
DISC frame .....	159, 172
Discover service .....	434
DiscoverReport.....	434
Discovery and Registration process, S-FSK .....	443
DL-CONNECT service .....	153
DL-CONNECT service,	433
DL-CONNECT.confirm .....	156
DL-CONNECT.indication.....	155
DL-CONNECT.request .....	154
DL-CONNECT.response .....	155
DL-DATA service .....	161

DL-Data services.....	432
DL-DATA services, connectionless.....	433
DL-DATA services, connection-oriented .....	433
DL-DATA.confirm .....	164
DL-DATA.indication .....	163
DL-DATA.request .....	162
DL-DISCONNECT service.....	157
DL-DISCONNECT.confirm .....	160
DL-DISCONNECT.indication .....	159
DL-DISCONNECT.request .....	158
DL-DISCONNECT.response .....	160
DL-GET_VALUE.confirm.....	192
DL-GET_VALUE.request .....	192
DL-INITIALIZE.confirm .....	191
DL-INITIALIZE.request .....	191
DL-LM_EVENT.indication .....	193
DLMS conformance .....	267
DLMS named variable .....	198
DLMS version number .....	267
DLMS/COSEM AL .....	52, 54, 55, 408
DLMS/COSEM AL, layer management services .....	202
DLMS/COSEM AL, protocol specification ..	203
DLMS/COSEM AL, service specification....	262
DLMS/COSEM AL, structure .....	194
DLMS/COSEM application layer	78, 194, 304, 407, 415
DLMS/COSEM aware .....	52
DLMS/COSEM client .....	57, 209
DLMS/COSEM client model .....	57
DLMS/COSEM conformance test process....	18
DLMS/COSEM security concept.....	204
DLMS/COSEM server model.....	54
DLMS/COSEM transport layer..	55, 56, 78, 415
DLMS/COSEM transport layer, state machine .....	85
DLMS/COSEM transport layer, TCP based..	85
DLMS/COSEM transport layer, TCP based, protocol specification.....	93
DLMS/COSEM transport layer, TCP based, service specification.....	86
DLMS/COSEM transport layer, UDP based .	80
DLMS/COSEM transport layer, UDP based, protocol specification.....	83
DLMS/COSEM transport layer, UDP based, service specification .....	80
DL-Reply services .....	432
DL-SET_VALUE.confirm .....	193
DL-SET_VALUE.request .....	192
DL-Update-Reply services .....	432
DM frame .....	173
DM response.....	173, 175, 178, 185
Domain parameters .....	217, 221, 232
Domain parameters, validity.....	232
Dynamic repeater .....	439
ECC_P256_Domain_Parameters .....	581
ECC_P384_Domain_Parameters .....	581
ECDSA algorithm .....	247
ECPoint .....	239
Elliptic curve .....	218, 220
Elliptic curve cryptography .....	217
Elliptic curve digital signature (ECDSA) .....	220
Encoding, A-XDR.....	312
Encoding, BER .....	312
Encryption .....	207, 211, 227, 249
Encryption key.....	228
Encryption, Mode of Operation .....	212
End entity .....	235
End entity certificate .....	242
End entity signature certificate .....	583
End-to-end security .....	52, 204
Ephemeral key .....	228
Ephemeral key agreement key pair .....	232
Ephemeral key pair.....	222
Ephemeral private key .....	221
Ephemeral public key.....	221
Ephemeral Unified Model .....	221, 231, 586
Error detection.....	85
Ethernet.....	56, 602
EventNotification service	51, 196, 197, 293, 335, 452, 454, 470
EventNotification service, 3-layer, CO, HDLC based profile .....	411
EventNotification service, TCP-UDP/IP based profile .....	421
exception-response .....	450
ExceptionResponse .....	323
Failure management .....	287
Failure_type .....	266
Fast FCS Implementation.....	187
Fast Synchronization .....	442
FCS calculation .....	187
FCS error .....	173, 186
FCS table generator .....	190
Field Element .....	219
fin segment .....	96
Fixed field .....	215
Flag field .....	167
Flow control.....	85
FORGOTTEN .....	446
Format type sub-field .....	167
Frame Check Sequence (FCS) .....	168
Frame format field .....	167
Frame length sub-field .....	167
Frame reject response (FRMR) .....	173
FRMR response.....	173, 186
FTP.....	56
Full-duplex .....	85, 151
Galois/Counter Mode .....	212, 213, 227
Gateway protocol.....	493, 494
General block transfer .....	201, 272, 332, 346
General protection .....	200, 201
General_Block_Transfer_Parameters .....	346
general-ciphering.....	201, 252, 255
general-ded-ciphering .....	201, 251
general-glo-ciphering .....	201, 251
GeneralizedTime .....	238
general-signing.....	201
generate_certificate_request method.....	244
generate_key_pair method.....	244

Generic communication profile .....	53
GET service .....	197, 277, 323
GET.confirm.....	280
GET.indication .....	280
GET.request.....	280
GET.response .....	280
Get_Data_Result.....	279
Get-Request.....	280
Get-Request-Next .....	324
GET-REQUEST-NEXT.....	278, 324, 326, 337
Get-Request-Normal.....	324
GET-REQUEST-NORMAL .	278, 279, 324, 337
Get-Request-With-List .....	324
GET-REQUEST-WITH-LIST	278, 279, 324, 337
Get-Response .....	280
GET-RESPONSE-LAST-BLOCK	278, 324, 337
Get-Response-Normal .....	324
GET-RESPONSE-NORMAL .....	278, 324, 337
GET-RESPONSE-ONE-BLOCK	278, 324, 325, 337
Get-Response-With-Datablock .....	324
Get-Response-With-List.....	324
GET-RESPONSE-WITH-LIST ....	278, 324, 337
Global broadcast encryption key, GBEK ...	228, 230, 231
Global key.....	228
Global unicast encryption key, GUEK	228, 230, 231
Graceful release.....	195
Half-duplex .....	59, 151
Hash function.....	210
Hash value.....	210
Hayes commands .....	69
HCS error .....	186
HDLC address.....	162
HDLC address field structure .....	168
HDLC addresses, special and reserved....	169
HDLC based data link layer .....	54, 55, 432
HDLC channel operation.....	175
HDLC channel states.....	174
HDLC extended addressing .....	168
HDLC frame .....	167
HDLC frame format type 3 .....	151, 167
HDLC non-operational modes .....	175
HDLC operational modes .....	175
HDLC parameter negotiation.....	176
HDLC protocol.....	433
HDLC, CALLING device physical address .	184
HDLC, command and response frames .....	171
HDLC, elements of the procedures.....	174
HDLC, exception recovery .....	185
HDLC, exchange of information frames .....	178
HDLC, exchanging data.....	178
HDLC, Multi- and broadcasting .....	181
HDLC, Response time-out .....	185
HDLC, segmentation .....	179
HDLC, selected repertoire .....	171
HDLC, sequence number.....	178
HDLC, transferring long MSDUs from the server to the client.....	180
HDLC, Window size considerations .....	179
Head End System.....	430, 493
Header Check Sequence (HCS) .....	168
High Level Security authentication .....	206
HLS authentication .....	259
HLS authentication mechanism 3, MD5 .....	259
HLS authentication mechanism 4, SHA-1 ..	259
HLS authentication mechanism 5, GMAC ..	260
HLS authentication mechanism 5, GMAC ) ..	259
HLS authentication mechanism 6, SHA-256 .....	259
HLS authentication mechanism 7, ECDSA ..	259, 261
HLS secret .....	207
HTTP .....	56
I frame, command and response .....	172
I_COMPLETE .....	163, 178
I_FIRST_FRAGMENT .....	163, 178, 180
I_FRAGMENT .....	163, 178, 180
I_LAST_FRAGMENT .....	163, 178, 180
Identification.....	18, 204, 416
Identification and addressing scheme .....	407, 408
Identification protocol specification.....	67
Identification service .....	65, 412
Identified_Key_Options .....	277
identified-key .....	230
IDENTIFY.request .....	66
IDENTIFY.response .....	66
Identifying service invocations .....	199
Idle HDLC channel state .....	175
IDLE sub-state .....	97
Implementation_Information .....	267
implementation-information .....	309
<i>import_certificate</i> method .....	243, 244
Inactivity time-out .....	186
Information field.....	168
Information security .....	13, 204
InformationReport service	51, 197, 302, 345, 470
InformationReport.request .....	345
informationReportRequest .....	345
Initial credit .....	436
Initialization vector.....	207, 212, 214, 215, 251
InitiateRequest APDU .....	499, 511
InitiateResponse APDU .....	500, 514
Initiator.....	95, 429, 447
Initiator L-SAP .....	447
Integer .....	218
Integrity .....	211
Intelligent Search Initiator process .....	434, 440
Interconnectivity .....	58
interface model.....	17
Inter-frame time-out .....	186
Internet Protocol.....	78, 415, 416
Inter-octet time-out .....	175
Interoperability .....	58
Invalid Frame .....	174
Invocation counter .....	254
Invocation field .....	215
Invoke_Id ...	199, 278, 281, 283, 326, 329, 330

Invoke-Id, long .....	199
Invoke-Id-And-Priority.....	420
IP network.....	78
IPv4 address .....	418
Kernel functional unit.....	307
Key agreement.....	217, 220, 227, 231
Key agreement, Ephemeral Unified Model.	221
Key agreement, One-Pass Diffie-Hellman..	222
Key agreement, Static Unified Model.....	223
Key derivation .....	221
Key Derivation Function.....	225
Key Encrypting Key, KEK...	213, 216, 228, 231
Key identification.....	230
Key information .....	229
Key pair generation .....	232
Key size .....	227
Key transport .....	18
Key wrapping .....	213, 227, 230
Key wrapping key .....	213
<i>key_agreement</i> method .....	231
Key_Info_Options .....	277
Key_set subfield.....	250
<i>key_transfer</i> method.....	231
key-ciphered-data.....	230
key-info.....	254
key-parameters .....	230
Last_Block	279, 282, 285, 297, 298, 300, 325, 328
last-block .....	359
Link Layer Address, M-Bus .....	464
Link Layer Address, wired M-Bus .....	465
Link Layer Address, wireless M-Bus.....	466
LLC addresses .....	447
LLC PDU format .....	165
LLC sublayer.....	151, 432
LLC sublayer, connectionless .....	432
LLC sublayer, HDLC based.....	432
LLC sublayer, protocol specification .....	165
LLC sublayer, state transition table .....	166
LN/SN data transfer service mapping .....	304
Local Network .....	45, 415
Local Network Access Points .....	430
Local_IP_Address .....	81
Local_or_Remote .....	266, 270
Local_TCP_Port .....	87
Local_UDP_Port.....	81
Local_wPort .....	81
Logical device .....	47, 54, 433
Logical Device Name.....	49
Logical Link Control.....	432
Logical name.....	198
Logical Name referencing .....	50, 196, 198
Long messages .....	452, 454
Long service parameters .....	199
Long_Invoke_Id.....	287, 288, 292
Long_M-Bus_Data_Header .....	462
Low Level Security authentication .....	206
Lower HDLC address.....	168
MAC address, HDLC .....	168
MAC address, S-FSK.....	446
MAC PDU.....	167
MAC sublayer.....	151, 432
MAC sublayer, data communication.....	165
MAC sublayer, physical layer services used by .....	164
MAC sublayer, protocol specification .....	167
MAC sublayer, state transition diagram .....	187
MA-CONNECT service.....	153
MA-CONNECT.confirm .....	156
MA-CONNECT.indication .....	155
MA-CONNECT.request .....	154
MA-CONNECT.response.....	155
MA-DATA service .....	161
MA-Data services .....	432
MA-DATA.confirm .....	164
MA-DATA.indication.....	163
MA-DATA.request .....	162
MA-DISCONNECT service .....	157
MA-DISCONNECT.confirm .....	160
MA-DISCONNECT.indication .....	159
MA-DISCONNECT.request.....	158
MA-DISCONNECT.response .....	160
Management Logical Device	48, 54, 55, 85, 293, 345, 447, 448, 468, 480, 483
Management Logical Device address .....	169
Manufacturer identification .....	466
Master key .....	228, 230, 231
Master/ Slave operation on the multi-drop bus .....	413
MA-Sync.indication .....	432
Max_Adr_MAC .....	437
MAX_NB_OF_RETRIES.....	176, 178, 186
Maximum information field length .....	187
MAXIMUM_INFORMATION_FIELD_LENGTH .....	176
M-Bus broadcast .....	466
M-Bus data header, long.....	468
M-Bus data header, short.....	468
M-Bus Link layer.....	458
M-Bus Physical layer .....	458
M-Bus profile, wired.....	456, 465
M-Bus profile, wireless.....	456
M-Bus TPDU format.....	467
M-Bus wrapper .....	464, 468
M-Bus, Client and server SAPs .....	468
M-Bus_Data_Header_Type .....	462
M-Bus-based transport layer .....	459
mechanism-name .....	309
Medium Access Control .....	432
Message Authentication Code .....	212
Message digest .....	210
Message integrity .....	213
Message security, client - server .....	208
Message security, end-to-end .....	209
Messaging.....	17
Messaging patterns .....	51
Meter installation .....	58
Metering equipment .....	54
METERING HDLC protocol .....	74
Method.....	17

Modelling .....	17
Multi- and broadcasting using UDP .....	82
Multi-drop configuration, 3-layer, CO, HDLC based profile .....	413
Multi-layer protection .....	201, 207, 258
Multiple references .....	198
Mutual authentication .....	206
N(S) sequence error .....	186
Naming .....	47
Nb_Tslot_For_New .....	437
Negotiated_DLMS_Version_Number .....	268
Negotiated_xDLMS_Context .....	267
Neighbourhood Network .....	45, 415
Neighbourhood Network Access Point .....	430
Never repeater .....	439
NEW .....	447
NEW and LOCKED state .....	443, 445, 446
NEW and UNLOCKED state .....	442, 443, 444, 446
New system .....	429
New system title .....	429
NIST Concatenation KDF .....	221
No TCP Connection .....	97
NO-BODY .....	444, 447
Non-basic frame format transparency .....	151
Nonce .....	215, 223
Non-critical certificate extension .....	235
None_M-Bus_Data_Header .....	462
Non-repudiation .....	210, 217
No-station address .....	169, 170
Notification_Body .....	292
NRM .....	174
NRM, Normal Response Mode .....	178
NSA Suite B .....	227, 581
OBJECT IDENTIFIER .....	238
Octet String .....	218
of Public Key Infrastructure .....	233
One-Pass Diffie-Hellman ...	221, 231, 255, 589
Order of bit and octet transmission .....	174
Organization .....	237
Organizational Unit .....	237
Originator .....	207
Originator_System_Title .....	259, 277
originator-system-title .....	226, 254
OSI model .....	45
OSI-style services .....	79
Other input .....	221
Other_Information .....	277
OtherInfo .....	225
other-information .....	254
Out of Band .....	233
P/F bit .....	170
Parameterized access .....	198
Parameterized_Access .....	294, 296, 297, 299, 300, 301
PartyUInfo .....	226
PartyVInfo .....	226
Passive opening .....	95
Password .....	206
P-Data services .....	432
PH Data transfer services .....	60
PH Layer management services .....	60
PH-ABORT .....	60
PH-ABORT service .....	63
PH-ABORT.confirm .....	63, 77
PH-ABORT.indication .....	63, 73, 77
PH-ABORT.request .....	63, 77
PH-ABORT.request/.confirm .....	73
PH-CONNECT .....	60
PH-CONNECT service .....	61
PH-CONNECT.confirm .....	62, 70, 77
PH-CONNECT.indication .....	62, 71, 77
PH-CONNECT.request .....	61, 69, 77
PH-DATA .....	60
PH-DATA service .....	62
PH-DATA.indication .....	63
PH-DATA.request .....	62
PH-DATA.request/.indication .....	72
PhL connection establishment/release services .....	60
PH-Layer service primitives .....	69, 77
Physical layer, data communications .....	69
Physical address .....	54, 55
Physical connection manager .....	412
Physical connection, setting up .....	64
Physical device .....	47, 54
Physical layer .....	54, 56, 76, 408, 432
Physical layer operation, procedures .....	64
Physical layer Protocol Data Unit .....	64
Physical layer services .....	59, 60, 69
Physical layer, disconnection .....	69
Physical layer, protocol specification .....	64
Physical layer, service definitions .....	61
Physical layer, service specification .....	60
Physical link, disconnecting .....	165
Physical link, setting up .....	164
Ping Service .....	435
PING service .....	434
Ping-no-response .....	436
Ping-system-title-nok .....	436
PKI architecture .....	234
Plaintext .....	211, 214, 227, 250
Point-to-multipoint .....	59, 151
Point-to-point .....	59, 151
Poll/final bit .....	171
Port number .....	56
Positive Acknowledgement with Retransmission .....	85, 96
Pre-established application association .....	268
Presentation layer .....	195
Priority 199, 278, 281, 283, 288, 326, 329, 330	
Private key .....	217, 220
Processing_Option .....	288, 292
Profile specific service parameters .....	409
Programming mode .....	75
Proposed_xDLMS_Context .....	267
proposed-conformance .....	315
proposed-dlms-version-number .....	315
Protection_Element .....	247, 274, 277, 346
Protocol connection parameters .....	266

Protocol identification service .....	58
Protocol mode E .....	74
<u>Protocol mode E, flow diagram .....</u>	75
Protocol specification, Phy layer .....	152
Protocol_Connection_Parameters .....	409, 419
Protocol_Parameters .....	294
protocol-version .....	309
P-Sync .....	432
Public attribute .....	198
Public Client	48, 54, 58, 85, 169, 447, 448, 468,
	480, 483
Public key .....	217, 220
Public key algorithm .....	210, 217, 231
Public key certificate .....	232
Public key infrastructure .....	232
Pull operation .....	51, 494
Push operation .....	51, 197, 209, 495
Push setup object .....	312
Random number generation .....	226
Raw_Data .....	279, 282, 285, 297, 298, 329
Read service .....	197, 295, 336
Read.confirm .....	298
Read.indication .....	298
Read.request .....	298
Read.response .....	298
Read_Data_Block_Access .....	294, 296, 297
Readout mode .....	75
readRequest .....	298
ReadRequest .....	337
readResponse .....	298
ReadResponse .....	298, 337, 338
Real-world IP networks .....	421
reason .....	310
Receive not ready .....	172
Receive ready .....	172
Reception_Threshold .....	437
Recipient .....	207
Recipient_System_Title .....	259, 277
recipient-system-title .....	226, 254
Reference model, S-FSK profile .....	431
Register service .....	434
REGISTERED and LOCKED state .....	445
REGISTERED and UNLOCKED state .....	445
Registered COSEM names .....	310
REGISTERED state .....	445, 446
Registered system .....	429
Remote_IP_Address .....	81
Remote_TCP_Port .....	87
Remote_UDP_Port .....	81
Remote_wPort .....	81
Repeater status .....	436, 439, 443
RepeaterCall service .....	434, 436
reply_to_HLS_authentication .....	206
Reporting system .....	429
Request_Type .....	278, 281, 283
Reserved wrapper port numbers .....	85, 94
reset_new_not_synchronized .....	446
Responder .....	95
responder-acse-requirements .....	316
Responding_AE_Qualifier .....	266
Responding-AE-Qualifier .....	316
Responding-AP-Title .....	316
responding-authentication-value .....	309, 316
Response time-out (TO_WAIT_RESP) .....	186
Response_Type .....	278, 281
response-allowed .....	313, 315, 317, 451
Result .....	266, 310, 325
Result (-) .....	298, 300
Result (+) .....	297, 300
Result Source-Diagnostic .....	310
Revision History .....	14
<b>RLRE APDU</b> .....	269
RLRE APDU, BER encoding .....	517
<b>RLRQ APDU</b> .....	269
RLRQ APDU, BER encoding .....	517
RNR frame .....	172
Root Certification Authority .....	234
Root-CA .....	234
Root-CA certificate .....	233
RR frame .....	172, 183, 185
Search Initiator Phase .....	440, 442, 446
search_initiator_threshold .....	442
search_initiator_time_out .....	442
Secret key algorithm .....	210
Secret keying material .....	220, 222, 223
Secure Hash Algorithm .....	227
Security algorithm ID .....	226
Security concept .....	204
Security context .....	50, 51, 204, 207
Security control byte .....	228, 250, 254
Security header .....	250
Security mechanism name .....	267
Security personalisation .....	244
Security policy .....	207, 247
Security setup .....	45, 233, 244
Security setup\ interface class .....	221
Security suite .....	220, 227
Security_Options .....	247, 274, 346
Security_Status .....	247, 274, 346
Security_Suite_Id .....	250
Segmentation .....	433
Segmentation bit .....	167, 180
Segmentation, HDLC .....	179
Segmentation, M-Bus .....	467
Selective access .....	45, 197, 287
Self_Descriptive .....	288, 292
Self-descriptive response .....	287
Semantic interoperability .....	58
SEND() function .....	82
SEND/RECEIVE state .....	97
Sender ACSE requirements .....	315
Sequence numbers .....	98, 172
Server .....	46, 47, 48, 49, 51, 52, 258
Server system title .....	316
Server_LLC_Address .....	409
Server_Max_Receive_PDU_Size .....	268
server_system_title .....	49
<b>Service Access Point</b> .....	57, 204
Service invocation .....	322, 348
Service mapping .....	407, 409

Service primitives .....	262
Service specification.....	153
Service specification, Phy layer .....	152
Service_Class278, 281, 283, 288, 292, 420, 451	
Service_Class == Unconfirmed .....	268
Service_Class parameter.....	268
Service-specific ciphering .....	251
Service-specific dedicated ciphering .....	251
Service-specific global ciphering .....	251
Set normal response mode .....	172
SET service .....	197, 280, 327
SET.confirm .....	282
SET.indication .....	282
SET.request.....	282
SET.response .....	282
SetMapperTable.....	202
SetMapperTable.request.....	303
Set-Request.....	282
SET-REQUEST-FIRST-BLOCK..	281, 327, 341
SET-REQUEST-FIRST-BLOCK-WITH-LIST .....	281, 327, 341
SET-REQUEST-LAST-BLOCK ...	281, 327, 341
Set-Request-Normal.....	327
SET-REQUEST-NORMAL..	281, 327, 341, 344
SET-REQUEST-ONE-BLOCK ...	281, 327, 341
Set-Request-With-Datablock.....	327
SET-REQUEST-WITH-FIRST-BLOCK .....	281
Set-Request-With-First-Datablock .....	327
Set-Request-With-List .....	327
SET-REQUEST-WITH-LIST	281, 327, 341, 344
Set-Request-With-List-And-With-First- Datablock.....	327
Set-Response .....	282
SET-RESPONSE-ACK-BLOCK ..	281, 327, 341
Set-Response-Datablock .....	327
SET-RESPONSE-LAST-BLOCK.	281, 327, 341
SET-RESPONSE-LAST-BLOCK-WITH-LIST .....	281, 327, 341
Set-Response-Last-Datablock.....	327
Set-Response-Normal .....	327
SET-RESPONSE-NORMAL .....	281, 327, 341
Set-Response-With-List.....	327
SET-RESPONSE-WITH-LIST .....	281, 327, 341
Setting up the data link .....	175
S-FSK PLC environment .....	48
Shared secret.....	220, 221, 222, 223
Short Name referencing.....	50, 196, 198
Short_M-Bus_Data_Header .....	462
signatureAlgorithm.....	236
signatureValue.....	236
SNRM command .....	175, 186
SNRM frame .....	155, 172, 185
Source address .....	167, 170
Source UDP .....	85
Source wPort .....	84
Special addresses .....	170
Start/stop transmission .....	175
Static key agreement key pair .....	232
Static key pair .....	222
Static private key .....	223
Static public key .....	223
Static Unified Model.....	221, 231, 593
Streaming.....	348
Sub_Tslot position .....	438
Sub-CA .....	235
Sub-CA certificate .....	233
Subject, public key certificate .....	233
Subordinate Authority .....	234
Sub-slot .....	429
<b>Supporting layer</b> .....	57
Supporting layer services .....	407, 409
Symmetric key .....	228
Symmetric key algorithm .....	18, 210, 211
Symmetric key block cipher .....	213
Synchronization .....	432
synchronization_locked .....	445
Syntactic interoperability .....	58
System integration .....	58
System title .....	47, 48
System_Title_Server .....	435
system-title.....	253
tbsCertificate .....	236
TCP Connected .....	97
TCP connection .....	94
TCP connection abort .....	96, 100
TCP connection closing .....	85, 99
TCP connection establishment .....	85, 98
TCP connection establishment by the server .....	421
TCP connection manager process .....	80, 86
TCP data communication .....	85
TCP disconnection .....	95
TCP packets .....	93
TCP-ABORT service .....	91
TCP-ABORT.indication .....	91
TCP-CONNECT .....	87
TCP-CONNECT.confirm .....	88
TCP-CONNECT.indication .....	87
TCP-CONNECT.request .....	87
TCP-CONNECT.response .....	87
TCP-DATA service .....	91, 101
TCP-DATA services .....	96
TCP-DATA.confirm .....	92
TCP-DATA.indication .....	92
TCP-DATA.request .....	91
TCP-DISCONNECT services .....	88
TCP-DISCONNECT.confirm .....	90
TCP-DISCONNECT.indication .....	89
TCP-DISCONNECT.request .....	88
TCP-DISCONNECT.response .....	89
TCP-UDP/IP based communication profile ..	53, 415
Third party .....	45, 52, 209, 258
Three-letter manufacturer ID .....	48
Three-way handshake .....	95, 98
<i>time_out_not_addressed</i> .....	435
Timeslot .....	429
TLS-Certificate .....	235

Transaction_Id .....	277	V(R) .....	173
transaction-id .....	254	V(S) .....	173
Transfer syntax .....	50, 446	Valid wPort numbers .....	82, 92
Transmission Control Protocol .....	85	Validity period .....	233
Transmission credits, M-Bus .....	471	Variable Access Specification .....	294
Transmission order and characteristics .....	64	Variable_Access_Specification .....	197, 297
Transparency .....	174	Variable_Name... 294, 296, 297, 299, 300, 301	
Transport layer address, M-Bus .....	464, 466	Variable-Access-Specification .....	296, 299, 301
Transporting.....	17	Virtual circuit .....	85
Transporting long messages, 3-layer, CO, HDLC based profile .....	413	Wide Area Network .....	45, 415
Transporting long messages, TCP-UDP/IP based profile .....	421	Window size .....	187
TriggerEventNotificationSending .....	412, 421	WINDOW_SIZE .....	176
TriggerEventNotificationSending service ..	294, 421, 452	wPort .....	417
Trust anchor.....	233, 243	wPort number .....	420
Trust model.....	233	Wrapped_Key_Options .....	277
Two-way alternate data transfer .....	151, 179	wrapped-key.....	230
UA frame .....	156, 173	Wrapper .....	79
UA response .....	172, 173, 175, 176, 178, 185	Wrapper header .....	83
UDP Datagram .....	82	Wrapper port .....	55, 56, 85
UDP-DATA service .....	80	Wrapper protocol data unit .....	83, 93
UDP-DATA.confirm.....	82	Wrapper sublayer .....	79, 83, 93, 417
UDP-DATA.indication .....	82	Wrapper sublayer, state transition diagram ..	97
UDP-DATA.request .....	81	Write service .....	197, 298, 340
UI frame.....	170, 178, 179, 181, 182, 183	Write.confirm .....	301
UI frame, sending from server to client.....	183	Write.indication.....	301
Unbalanced connection-mode .....	151	Write.request.....	300
UNC basic repertoire .....	171	Write.response .....	301
Uncompressed form, ECC public key .....	239	Write_Data_Block_Access .....	294, 299
Unconfirmed service .....	196	writeRequest .....	300
Unconfirmed service invocations.....	322	WriteRequest .....	341
UnconfirmedWrite service .....	197, 301, 344	writeResponse .....	301
UnconfirmedWrite.indication .....	302	WriteResponse .....	300, 341
UnconfirmedWrite.request .....	302	X.509 v3 Certificate .....	235
unconfirmedWriteRequest.....	302	xDLMS ASE .....	52, 194, 196
Unified service .....	286	xDLMS conformance block .....	202
Unilateral authentication .....	206	xDLMS context .....	50, 195
Unnumbered information (UI) command and response.....	173	xDLMS initiate service .....	196
Unsolicited service .....	197	xDLMS InitiateRequest .....	228, 267
Upper HDLC address.....	168	xDLMS InitiateResponse .....	228
User Datagram Protocol .....	80	xDLMS procedures .....	203
User_Information.....	268	xDLMS services, LN referencing .....	303
user-information .....	309, 310	xDLMS version .....	201
		XML document .....	195
		XML schema .....	312, 384
		XML Schema Definitions Language .....	384
		XX-DISCONNECT.request .....	271