

Green Book Edition 8.1

TECHNICAL REPORT

Companion Specification
for Energy Metering

DLMS/COSEM

**Architecture
and Protocols**

DLMS User Association



CONTENTS

Foreword.....	25
1 Scope	31
2 Referenced documents (see also the Bibliography).....	32
3 Terms, Definitions and Abbreviations	35
3.1 General DLMS/COSEM definitions	35
3.2 Definitions related to cryptographic security	37
3.3 Definitions and abbreviations related to the Galois/Counter Mode	46
3.4 General abbreviations	47
3.5 Symbols related to the Galois/Counter Mode	51
3.6 Symbols related the ECDSA algorithm.....	52
3.7 Symbols related to the key agreement algorithms.....	52
3.8 Abbreviations related to the DLMS/COSEM M-Bus communication profile	52
4 Information exchange in DLMS/COSEM.....	53
4.1 General	53
4.2 Communication model.....	53
4.3 Naming and addressing.....	54
4.3.1 General.....	54
4.3.2 Naming	55
4.3.3 Addressing.....	55
4.3.4 System title	56
4.3.5 Logical Device Name	57
4.3.6 Client user identification.....	57
4.4 Connection oriented operation.....	57
4.5 Application associations	57
4.5.1 General.....	57
4.5.2 Application context.....	58
4.5.3 Authentication	58
4.5.4 xDLMS context	59
4.5.5 Security context	59
4.5.6 Access rights	59
4.6 Messaging patterns	59
4.7 Data exchange between third parties and DLMS/COSEM servers.....	60
4.8 Communication profiles	60
4.9 Model of a DLMS/COSEM metering system.....	61
4.10 Model of DLMS/COSEM servers.....	62
4.11 Model of a DLMS/COSEM client	63
4.12 Interoperability and interconnectivity in DLMS/COSEM	64
4.13 Ensuring interconnectivity: the protocol identification service	65
4.14 System integration and meter installation	65
5 Physical layer services and procedures for connection-oriented asynchronous data exchange	66
5.1 Overview	66
5.2 Service specification	67
5.2.1 List of services	67

5.2.2	Use of the physical layer services	67
5.2.3	Service definitions.....	68
5.2.3.1	The PH-CONNECT service.....	68
5.2.3.1.1	PH-CONNECT.request.....	68
5.2.3.1.2	PH-CONNECT.indication.....	68
5.2.3.1.3	PH-CONNECT.confirm	69
5.2.3.2	The PH-DATA service	69
5.2.3.2.1	PH-DATA.request.....	69
5.2.3.2.2	PH-DATA.indication	70
5.2.3.3	The PH-ABORT service.....	70
5.2.3.3.1	PH-ABORT.request.....	70
5.2.3.3.2	PH-ABORT.confirm	70
5.2.3.3.3	PH-ABORT.indication.....	70
5.3	Protocol specification	71
5.3.1	Physical layer protocol data unit.....	71
5.3.2	Transmission order and characteristics	71
5.3.3	Physical layer operation – description of the procedures	71
5.3.3.1	General	71
5.3.3.2	Setting up a physical connection	72
5.3.3.3	The Identification service.....	73
5.3.3.3.1	General.....	73
5.3.3.3.2	Identification service specification	73
5.3.3.3.2.1	IDENTIFY.request	73
5.3.3.3.2.2	IDENTIFY.response	73
5.3.3.3.3	Identification service protocol specification.....	74
5.3.3.4	Data transfer	76
5.3.3.5	Disconnection of an existing physical connection	76
5.4	Example: PhL service primitives and Hayes commands.....	76
5.4.1	General.....	76
5.4.2	Physical layer services and related message exchanges.....	76
5.4.2.1	PH-CONNECT.request	76
5.4.2.2	PH-CONNECT.confirm	77
5.4.2.3	PH-CONNECT.indication	78
5.4.2.4	PH-DATA.request / .indication	79
5.4.2.5	PH-ABORT.request / .confirm.....	80
5.4.2.6	PH-ABORT.indication	80
6	Direct Local Connection (excerpt)	81
6.1	Introduction	81
6.2	METERING HDLC protocol using protocol mode E for direct local data exchange	81
6.3	Overview	81
6.4	Readout mode and programming mode	82
6.5	Key to protocol mode E flow diagram	82
6.6	Physical layer – Introduction.....	83
6.7	Physical layer primitives	84
6.8	Data link layer	84
7	DLMS/COSEM transport layer for IP networks	85
7.1	Scope.....	85
7.2	Overview	85

7.3	The DLMS/COSEM connection-less, UDP-based transport layer	87
7.3.1	General.....	87
7.3.2	Service specification for the DLMS/COSEM UDP-based transport layer	87
7.3.2.1	General	87
7.3.2.2	The UDP-DATA service	88
7.3.2.2.1	UDP-DATA.request	88
7.3.2.2.2	UDP-DATA.indication	88
7.3.2.2.3	UDP-DATA.confirm	89
7.3.3	Protocol specification for the DLMS/COSEM UDP-based transport layer	89
7.3.3.1	General	89
7.3.3.2	The wrapper protocol data unit (WPDU)	90
7.3.3.3	The DLMS/COSEM UDP-based transport layer protocol data unit	90
7.3.3.4	Reserved wrapper port numbers (wPort)	91
7.3.3.5	Protocol state machine	91
7.4	The DLMS/COSEM connection-oriented, TCP-based transport layer	92
7.4.1	General.....	92
7.4.2	Service specification for the DLMS/COSEM TCP-based transport layer	92
7.4.2.1	General	92
7.4.2.2	The TCP-CONNECT service.....	93
7.4.2.2.1	TCP-CONNECT.request.....	93
7.4.2.2.2	TCP-CONNECT.indication.....	93
7.4.2.2.3	TCP-CONNECT.response	94
7.4.2.2.4	TCP-CONNECT.confirm	94
7.4.2.3	The TCP-DISCONNECT service	95
7.4.2.3.1	TCP-DISCONNECT.request	95
7.4.2.3.2	TCP-DISCONNECT.indication	95
7.4.2.3.3	TCP-DISCONNECT.response	96
7.4.2.3.4	TCP-DISCONNECT.confirm	96
7.4.2.4	The TCP-ABORT service.....	97
7.4.2.4.1	TCP-ABORT.indication	97
7.4.2.5	The TCP-DATA service	98
7.4.2.5.1	TCP-DATA.request.....	98
7.4.2.5.2	TCP-DATA.indication	98
7.4.2.5.3	TCP-DATA.confirm	99
7.4.3	Protocol specification for the DLMS/COSEM TCP-based transport layer	100
7.4.3.1	General	100
7.4.3.2	The wrapper protocol data unit (WPDU)	100
7.4.3.3	The DLMS/COSEM TCP-based transport layer protocol data unit	100
7.4.3.4	Reserved wrapper port numbers.....	101
7.4.3.5	Definition of the procedures.....	101
7.4.3.5.1	TCP connection.....	101
7.4.3.5.2	TCP disconnection	102
7.4.3.5.3	TCP connection abort.....	102
7.4.3.5.4	Data transfer using the TCP-DATA service.....	103

	7.4.3.5.5 High-level state transition diagram of the wrapper sublayer	103
7.5	Converting OSI-style TL services to and from RFC-style TCP function calls ..	104
7.5.1	Transport layer and TCP connection establishment.....	104
7.5.2	Closing a transport layer and a TCP connection	106
7.5.3	TCP connection abort	106
7.5.4	Data transfer using the TCP-DATA service.....	107
8	Data Link Layer using the HDLC protocol.....	110
8.1	Overview	110
8.1.1	General.....	110
8.1.2	Structure of the data link layer	110
8.1.3	Specification method.....	111
8.2	Service specification	112
8.2.1	General.....	112
8.2.2	Setting up the data link connection: the DL-CONNECT and MA-CONNECT services.....	112
8.2.2.1	Overview	112
8.2.2.2	DL-CONNECT.request and MA-CONNECT.request	113
8.2.2.3	DL-CONNECT.indication and MA-CONNECT.indication.....	114
8.2.2.4	DL-CONNECT.response and MA-CONNECT.response	114
8.2.2.5	DL-CONNECT.confirm and MA-CONNECT.confirm	115
8.2.3	Disconnecting the data link connection: the DL-DISCONNECT and MA-DISCONNECT services.....	116
8.2.3.1	Overview	116
8.2.3.2	DL-DISCONNECT.request and MA-DISCONNECT.request.....	117
8.2.3.3	DL-DISCONNECT.indication and MA-DISCONNECT.indication	118
8.2.3.4	DL-DISCONNECT.response and MA-DISCONNECT.response	119
8.2.3.5	DL-DISCONNECT.confirm and MA-DISCONNECT.confirm	119
8.2.4	Data transfer: the DL-DATA and MA-DATA services	120
8.2.4.1	Overview	120
8.2.4.2	DL-DATA.request and MA-DATA.request.....	121
8.2.4.3	DL-DATA.indication and MA-DATA.indication	122
8.2.4.4	DL-DATA.confirm and MA-DATA.confirm	123
8.2.5	Physical layer services used by the MAC sublayer	123
8.2.5.1	Overview	123
8.2.5.2	Setting up a physical link.....	123
8.2.5.3	Disconnecting the physical link.....	124
8.2.5.4	Data transfer	124
8.3	Protocol specification for the LLC sublayer.....	124
8.3.1	Role of the LLC sublayer.....	124
8.3.2	LLC PDU format.....	124
8.3.3	State transition tables for the LLC sublayer	125
8.4	Protocol specification for the MAC sublayer.....	126
8.4.1	The MAC PDU and the HDLC frame.....	126
8.4.1.1	HDLC frame format type 3	126
8.4.1.2	Flag field	126
8.4.1.3	Frame format field	126
8.4.1.4	Destination and source address fields	126

8.4.1.5 Control field.....	126
8.4.1.6 Header check sequence (HCS) field	127
8.4.1.7 Information field	127
8.4.1.8 Frame check sequence (FCS) field.....	127
8.4.2 MAC addressing.....	127
8.4.2.1 Use of extended addressing	127
8.4.2.2 Address field structure.....	127
8.4.2.3 Reserved special HDLC addresses.....	128
8.4.2.4 Handling special addresses	129
8.4.2.5 Handling inopportune address lengths in the server	129
8.4.3 Command and response frames.....	130
8.4.3.1 Selected repertoire and control field format	130
8.4.3.2 Information transfer command and response	130
8.4.3.3 Receive ready (RR) command and response	131
8.4.3.4 Receive not ready (RNR) command and response.....	131
8.4.3.5 Set normal response mode (SNRM) command.....	131
8.4.3.6 Disconnect (DISC) command.....	131
8.4.3.7 Unnumbered acknowledge (UA) response	132
8.4.3.8 Disconnected mode (DM) response	132
8.4.3.9 Frame reject (FRMR) response	132
8.4.3.10 Unnumbered information (UI) command and response	132
8.4.4 Elements of the procedures.....	132
8.4.4.1 Overview	132
8.4.4.2 Transmission considerations	133
8.4.4.2.1 Transparency	133
8.4.4.2.2 Order of bit and octet transmission.....	133
8.4.4.2.3 Invalid frame	133
8.4.4.3 HDLC channel states.....	133
8.4.4.3.1 Active HDLC channel state.....	133
8.4.4.3.2 Abort sequence	133
8.4.4.3.3 Start/stop transmission inter-octet time-out	134
8.4.4.3.4 Idle HDLC channel state	134
8.4.5 HDLC channel operation – Description of the procedures.....	134
8.4.5.1 General	134
8.4.5.2 Data station characteristics	134
8.4.5.3 Procedures for setting up and disconnecting the data link	134
8.4.5.3.1 Setting up the data link	134
8.4.5.3.2 HDLC parameter negotiation during the connection phase.....	135
8.4.5.3.3 Disconnecting the data link	136
8.4.5.4 Procedures for data exchange.....	137
8.4.5.4.1 General.....	137
8.4.5.4.2 Exchange of information frames	137
8.4.5.4.3 Window size considerations	138
8.4.5.4.4 Segmentation	138
8.4.5.4.5 Transferring long MSDUs from the server to the client	138
8.4.5.4.6 Multi- and broadcasting	139
8.4.5.4.7 Sending an UI frame from the server to the client....	141

8.4.5.4.8	Handling the CALLING device physical address	143
8.4.5.5	Exception recovery.....	144
8.4.5.5.1	Response time-out	144
8.4.5.5.2	FCS and HCS error	144
8.4.5.5.3	N(S) sequence error.....	144
8.4.5.5.4	Command/response frame rejection	144
8.4.5.5.5	Busy.....	144
8.4.5.6	Time-outs and other MAC sublayer parameters	144
8.4.5.6.1	Time-out 1: Response time-out (TO_WAIT_RESP)..	144
8.4.5.6.2	Layer Parameter 1: Maximum number of retries (MAX_NB_OF_RETRIES).....	145
8.4.5.6.3	Time-out 2: Inactivity time-out	145
8.4.5.6.4	Time-out 3: Inter-frame time-out.....	145
8.4.5.6.5	Maximum information field length	145
8.4.5.6.6	Window size.....	145
8.4.5.7	State transition diagram for the server MAC sublayer	145
8.5	FCS calculation	146
8.5.1	Test sequence for the FCS calculation	146
8.5.2	Fast frame check sequence (FCS) implementation.....	146
8.5.3	16-bit FCS computation method	146
8.5.4	FCS table generator.....	147
8.6	Data link layer management services	149
8.6.1	Overview.....	149
8.6.2	Data link layer management service definitions	149
8.6.2.1	DL-INITIALIZE.request	149
8.6.2.2	DL-INITIALIZE.confirm	149
8.6.2.3	DL-GET_VALUE.request	150
8.6.2.4	DL-GET_VALUE.confirm	150
8.6.2.5	DL-SET_VALUE.request.....	150
8.6.2.6	DL-SET_VALUE.confirm	151
8.6.2.7	DL-LM_EVENT.indication	151
9	DLMS/COSEM application layer	152
9.1	DLMS/COSEM application layer main features	152
9.1.1	General	152
9.1.2	DLMS/COSEM application layer structure	152
9.1.3	The Association Control Service Element, ACSE	153
9.1.4	The xDLMS application service element	154
9.1.4.1	Overview	154
9.1.4.2	The xDLMS initiate service	154
9.1.4.3	COSEM object related xDLMS services	154
9.1.4.3.1	General	154
9.1.4.3.2	xDLMS services used by the client with LN referencing	155
9.1.4.3.3	xDLMS services used by the client with SN referencing	155
9.1.4.3.4	Unsolicited services	155
9.1.4.3.5	Selective access	155
9.1.4.3.6	Multiple references	156
9.1.4.3.7	Attribute_0 referencing	156
9.1.4.4	Additional mechanisms	156

9.1.4.4.1	Overview	156
9.1.4.4.2	Referencing methods and service mapping	156
9.1.4.4.3	Identification of service invocations: the Invoke_Id parameter	157
9.1.4.4.4	Priority handling	157
9.1.4.4.5	Transferring long messages	157
9.1.4.4.6	Composable xDLMS messages	158
9.1.4.4.7	Compression and decompression	158
9.1.4.4.8	General protection	158
9.1.4.4.9	General block transfer (GBT)	159
9.1.4.5	Additional data types	159
9.1.4.6	xDLMS version number	159
9.1.4.7	xDLMS conformance block	159
9.1.4.8	Maximum PDU size	159
9.1.5	Layer management services	160
9.1.6	Summary of DLMS/COSEM application layer services	160
9.1.7	DLMS/COSEM application layer protocols	161
9.2	Information security in DLMS/COSEM	161
9.2.1	Overview	161
9.2.2	The DLMS/COSEM security concept	162
9.2.2.1	Overview	162
9.2.2.2	Identification and authentication	162
9.2.2.2.1	Identification	162
9.2.2.2.2	Authentication mechanisms	162
9.2.2.2.2.1	Overview	162
9.2.2.2.2.2	No security (Lowest Level Security) authentication	163
9.2.2.2.2.3	Low Level Security (LLS) authentication	164
9.2.2.2.2.4	High Level Security (HLS) authentication	164
9.2.2.3	Security context	165
9.2.2.4	Access rights	165
9.2.2.5	Application layer message security	165
9.2.2.6	COSEM data security	168
9.2.3	Cryptographic algorithms	168
9.2.3.1	Overview	168
9.2.3.2	Hash function	168
9.2.3.3	Symmetric key algorithms	169
9.2.3.3.1	General	169
9.2.3.3.2	Encryption and decryption	169
9.2.3.3.3	Advanced Encryption Standard	170
9.2.3.3.4	Encryption Modes of Operation	170
9.2.3.3.5	Message Authentication Code	170
9.2.3.3.6	Key wrapping	171
9.2.3.3.7	Galois/Counter Mode	171
9.2.3.3.7.1	General	171
9.2.3.3.7.2	GCM functions	172
9.2.3.3.7.3	The initialization vector, <i>IV</i>	173
9.2.3.3.7.4	The encryption key, <i>EK</i>	174

9.2.3.3.7.5	The authentication key, <i>AK</i>	174
9.2.3.3.7.6	Length of the authentication tag	174
9.2.3.3.8	AES key wrap.....	174
9.2.3.4	Public key algorithms	175
9.2.3.4.1	General.....	175
9.2.3.4.2	Elliptic curve cryptography	175
9.2.3.4.2.1	General.....	175
9.2.3.4.2.2	NIST recommended elliptic curves	176
9.2.3.4.3	Data conversions	176
9.2.3.4.3.1	Overview	176
9.2.3.4.3.2	Conversion between Bit Strings and Octet Strings (BS2OS).....	176
9.2.3.4.3.3	Conversion between Octet Strings and Bit Strings (OS2BS).....	176
9.2.3.4.3.4	Conversion between Integers and Octet Strings (I2OS)	176
9.2.3.4.3.5	Conversion between Octet Strings and Integers (OS2I)	177
9.2.3.4.3.6	Conversion between Field Elements and Octet Strings (FE2OS)	177
9.2.3.4.3.7	Conversion between Octet Strings and Field Elements (OS2FE)	177
9.2.3.4.4	Digital signature	177
9.2.3.4.5	Elliptic curve digital signature (ECDSA)	178
9.2.3.4.6	Key agreement	179
9.2.3.4.6.1	Overview	179
9.2.3.4.6.2	The Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme	179
9.2.3.4.6.3	The One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme	180
9.2.3.4.6.4	The Static Unified Model C(0e, 2s, ECC CDH) scheme	181
9.2.3.4.6.5	Key Derivation Function – The NIST Concatenation KDF	183
9.2.3.5	Random number generation	184
9.2.3.6	Compression	184
9.2.3.7	Security suite	184
9.2.4	Cryptographic keys – overview	185
9.2.5	Key used with symmetric key algorithms	186
9.2.5.1	Symmetric keys types	186
9.2.5.2	Key information with general-ciphering APDU and data protection	187
9.2.5.3	Key identification	188
9.2.5.4	Key wrapping	188
9.2.5.5	Key agreement	189
9.2.5.6	Symmetric key cryptoperiods	189
9.2.6	Keys used with public key algorithms	189
9.2.6.1	Overview	189
9.2.6.2	Key pair generation	190
9.2.6.3	Public key certificates and infrastructure	190
9.2.6.3.1	Overview	190

9.2.6.3.2	Trust model	191
9.2.6.3.3	PKI architecture – informative	192
9.2.6.3.3.1	General	192
9.2.6.3.3.2	Root-CA	192
9.2.6.3.3.3	Sub-CA	193
9.2.6.3.3.4	End entities	193
9.2.6.4	Certificate and certificate extension profile	193
9.2.6.4.1	General	193
9.2.6.4.2	The X.509 v3 Certificate	193
9.2.6.4.3	tbsCertificate	194
9.2.6.4.3.1	Overview	194
9.2.6.4.3.2	Serial number	195
9.2.6.4.3.3	Issuer and Subject	195
9.2.6.4.3.4	Validity period	196
9.2.6.4.3.5	SubjectPublicKeyInfo	196
9.2.6.4.3.6	Subject Unique ID	197
9.2.6.4.4	Certificate extensions	197
9.2.6.4.4.1	Overview	197
9.2.6.4.4.2	Authority Key Identifier	198
9.2.6.4.4.3	SubjectKeyIdentifier	198
9.2.6.4.4.4	KeyUsage	198
9.2.6.4.4.5	CertificatePolicies	198
9.2.6.4.4.6	SubjectAltNames	198
9.2.6.4.4.7	IssuerAltName	199
9.2.6.4.4.8	Basic constraints	199
9.2.6.4.4.9	Extended Key Usage	200
9.2.6.4.4.10	cRLDistributionPoints	200
9.2.6.4.4.11	Other extensions	200
9.2.6.5	Suite B end entity certificate types to be supported by DLMS/COSEM servers	200
9.2.6.6	Management of certificates	201
9.2.6.6.1	Overview	201
9.2.6.6.2	Provisioning servers with trust anchors	201
9.2.6.6.3	Provisioning the server with further CA certificates ..	201
9.2.6.6.4	Security personalisation of the server	201
9.2.6.6.5	Provisioning servers with certificates of clients and third parties	203
9.2.6.6.6	Provisioning clients and third parties with certificates of servers	203
9.2.6.6.7	Certificate removal from the server	204
9.2.7	Applying cryptographic protection	204
9.2.7.1	Overview	204
9.2.7.2	Protecting xDLMS APDUs	205
9.2.7.2.1	Overview	205
9.2.7.2.2	Security policy and access rights values	205
9.2.7.2.3	Ciphered xDLMS APDUs	206
9.2.7.2.4	Encryption, authentication and compression	207
9.2.7.2.4.1	Overview	207

	9.2.7.2.4.2	The security header	208
	9.2.7.2.4.3	Plaintext and Additional Authenticated Data	209
	9.2.7.2.4.4	Encryption key and authentication key	209
	9.2.7.2.4.5	Initialization vector	209
	9.2.7.2.4.6	Service-specific ciphering xDLMS APDUs	209
	9.2.7.2.4.7	The general-glo-ciphering and general-ded-ciphering xDLMS APDUs	211
	9.2.7.2.4.8	The general-ciphering APDU	212
	9.2.7.2.4.9	Use of the fields of the ciphering xDLMS APDUs	212
	9.2.7.2.4.10	Encoding example: global-get-request xDLMS APDU	213
	9.2.7.2.5	Digital signature	217
	9.2.7.3	Multi-layer protection by multiple parties	217
	9.2.7.4	HLS authentication mechanisms	218
	9.2.7.5	Protecting COSEM data	220
9.3		DLMS/COSEM application layer service specification	221
9.3.1		Service primitives and parameters	221
9.3.2		The COSEM-OPEN service	223
9.3.3		The COSEM-RELEASE service	228
9.3.4		The COSEM-ABORT service	230
9.3.5		Protection and general block transfer parameters	230
9.3.6		The GET service	236
9.3.7		The SET service	238
9.3.8		The ACTION service	241
9.3.9		The ACCESS service	245
9.3.9.1		Overview – Main features	245
9.3.9.1.1		General	245
9.3.9.1.2		Unified WITH-LIST service to improve efficiency	245
9.3.9.1.3		Specific variants for selective access	245
9.3.9.1.4		Long_Invoke_Id parameter	246
9.3.9.1.5		Self-descriptive responses	246
9.3.9.1.6		Failure management	246
9.3.9.1.7		Time stamp as a service parameter	246
9.3.9.1.8		Presence of data in service primitives	246
9.3.9.2		Service specification	246
9.3.10		The DataNotification service	251
9.3.11		The EventNotification service	251
9.3.12		The TriggerEventNotificationSending service	252
9.3.13		Variable access specification	253
9.3.14		The Read service	254
9.3.15		The Write service	257
9.3.16		The UnconfirmedWrite service	259
9.3.17		The InformationReport service	261
9.3.18		Client side layer management services: the SetMapperTable.request	261
9.3.19		Summary of services and LN/SN data transfer service mapping	262
9.4		DLMS/COSEM application layer protocol specification	263
9.4.1		The control function (CF)	263

9.4.1.1	State definitions of the client side control function	263
9.4.1.2	State definitions of the server side control function	264
9.4.2	The ACSE services and APDUs	265
9.4.2.1	ACSE functional units, services and service parameters.....	265
9.4.2.2	Registered COSEM names	268
9.4.2.2.1	General.....	268
9.4.2.2.2	The COSEM application context.....	268
9.4.2.2.3	The COSEM authentication mechanism name	269
9.4.2.2.4	Cryptographic algorithm ID-s.....	270
9.4.3	APDU encoding rules	270
9.4.3.1	Encoding of the ACSE APDUs.....	270
9.4.3.2	Encoding of the xDLMS APDUs.....	270
9.4.3.3	XML	270
9.4.4	Protocol for application association establishment	271
9.4.4.1	Protocol for the establishment of confirmed application associations	271
9.4.4.2	Repeated COSEM-OPEN service invocations.....	274
9.4.4.3	Establishment of unconfirmed application associations.....	275
9.4.4.4	Pre-established application associations	275
9.4.5	Protocol for application association release	275
9.4.5.1	Overview	275
9.4.5.2	Graceful release of an application association	275
9.4.5.3	Non-graceful release of an application association	278
9.4.6	Protocol for the data transfer services	278
9.4.6.1	Negotiation of services and options – the conformance block	278
9.4.6.2	Confirmed and unconfirmed xDLMS service invocations	279
9.4.6.3	Protocol for the GET service.....	280
9.4.6.4	Protocol for the SET service	283
9.4.6.5	Protocol for the ACTION service.....	286
9.4.6.6	Protocol for the ACCESS service.....	288
9.4.6.7	Protocol of the DataNotification service	289
9.4.6.8	Protocol for the EventNotification service	290
9.4.6.9	Protocol for the Read service	290
9.4.6.10	Protocol for the Write service	294
9.4.6.11	Protocol for the UnconfirmedWrite service	298
9.4.6.12	Protocol for the InformationReport service	299
9.4.6.13	Protocol of general block transfer mechanism	299
9.5	Abstract syntax of COSEM APDUs	310
9.6	COSEM APDU XML schema.....	324
9.6.1	General	324
9.6.2	XML Schema	324
10	Using the DLMS/COSEM application layer in various communications profiles	346
10.1	Communication profile specific elements	346
10.1.1	General.....	346
10.1.2	Targeted communication environments	346
10.1.3	The structure of the profile	346
10.1.4	Identification and addressing schemes	346
10.1.5	Supporting layer services and service mapping	346

10.1.6	Communication profile specific parameters of the DLMS/COSEM AL services	347
10.1.7	Specific considerations / constraints using certain services within a given profile	347
10.2	The 3-layer, connection-oriented, HDLC based communication profile.....	347
10.2.1	Targeted communication environments	347
10.2.2	The structure of the profile	347
10.2.3	Identification and addressing scheme.....	347
10.2.4	Supporting layer services and service mapping.....	348
10.2.5	Communication profile specific service parameters of the DLMS/COSEM AL services	349
10.2.6	Specific considerations / constraints	349
10.2.6.1	Confirmed and unconfirmed AAs and data transfer service invocations, frame types used	349
10.2.6.2	Correspondence between AAs and data link layer connections, releasing AAs	350
10.2.6.3	Service parameters of the COSEM-OPEN / -RELEASE / -ABORT services.....	350
10.2.6.4	EventNotification service and protocol.....	351
10.2.6.5	Transporting long messages.....	352
10.2.6.6	Supporting multi-drop configurations	353
10.3	The TCP-UDP/IP based communication profiles (COSEM_on_IP)	355
10.3.1	Targeted communication environments	355
10.3.2	The structure of the profile(s)	355
10.3.3	Identification and addressing scheme.....	356
10.3.4	Supporting layer services and service mapping	358
10.3.5	Communication profile specific service parameters of the DLMS/COSEM AL services	359
10.3.6	Specific considerations / constraints	359
10.3.6.1	Confirmed and unconfirmed AAs and data transfer service invocations, packet types used	359
10.3.6.2	Releasing application associations: using RLRQ/RLRE is mandatory	360
10.3.6.3	Service parameters of the COSEM-OPEN / -RELEASE / -ABORT services.....	361
10.3.6.4	xDLMS request/response type services	361
10.3.6.5	The EventNotification Service and the TriggerEventNotificationSending service	361
10.3.6.6	Transporting long messages.....	361
10.3.6.7	Allowing COSEM servers to establish the TCP connection	361
10.3.6.8	The COSEM TCP-UDP/IP profile and real-world IP networks....	361
10.4	The S-FSK PLC profile	362
10.4.1	Terms and definitions relevant for the S-FSK PLC profile	362
10.4.2	Abbreviations relevant for the S-FSK PLC profile	362
10.4.3	Targeted communication environments	363
10.4.4	Reference model.....	364
10.4.4.1	Introduction	364
10.4.4.2	The physical layer (PhL).....	364
10.4.4.3	The data link layer	365
10.4.4.3.1	General	365

10.4.4.3.2 The MAC sublayer.....	365
10.4.4.3.3 The connectionless LLC sublayer.....	365
10.4.4.3.4 The HDLC based LLC sublayer	365
10.4.4.3.5 Co-existence of the connectionless and the HDLC based LLC sublayers.....	366
10.4.4.4 The application layer (AL).....	366
10.4.4.5 The application process (AP).....	366
10.4.5 The Configuration Initiation Application Service Element (CIASE)	366
10.4.5.1 Overview	367
10.4.5.2 The Discover service	367
10.4.5.3 The Register service	367
10.4.5.4 The Ping Service	368
10.4.5.5 The RepeaterCall service	369
10.4.5.6 The ClearAlarm service	371
10.4.5.7 The Intelligent Search Initiator process.....	373
10.4.5.7.1 Introduction	373
10.4.5.7.2 Operation	373
10.4.5.7.2.1 Flow chart	373
10.4.5.7.2.2 Process parameters	373
10.4.5.7.2.3 Search Initiator Phase.....	374
10.4.5.7.2.4 Check Initiator Phase	375
10.4.5.7.2.5 Remarks	376
10.4.5.8 The Discovery and Registration process,.....	376
10.4.5.9 Abstract and transfer syntax	379
10.4.6 Addressing.....	379
10.4.6.1 General	379
10.4.6.2 IEC 61334-5-1 MAC addresses	379
10.4.6.3 Reserved special LLC addresses.....	379
10.4.6.3.1 General.....	379
10.4.6.3.2 Reserved addresses for the IEC 61334-4-32 LLC sublayer	379
10.4.6.3.3 Reserved addresses for the HDLC based LLC sublayer	380
10.4.6.4 Source and destination APs and addresses of CI-PDUs	380
10.4.7 Specific considerations / constraints for the IEC 61334-4-32 LLC sublayer based profile	381
10.4.7.1 Establishing application associations.....	381
10.4.7.2 Application association types, confirmed and unconfirmed xDLMS services.....	382
10.4.7.3 xDLMS request/response type services	383
10.4.7.4 Releasing application associations	383
10.4.7.5 Service parameters of the COSEM-OPEN / -RELEASE / -ABORT services.....	384
10.4.7.6 The EventNotification service and the TriggerEventNotificationSending service	384
10.4.7.7 Transporting long messages.....	384
10.4.7.8 Broadcasting	384
10.4.8 Specific considerations / constraints for the HDLC LLC sublayer based profile	385
10.4.8.1 Establishing application associations.....	385

10.4.8.2 Application association types, confirmed and unconfirmed xDLMS services.....	386
10.4.8.3 xDLMS request/response type services	386
10.4.8.4 Correspondence between AAs and data link layer connections, releasing AAs	386
10.4.8.5 Service parameters of the COSEM-OPEN/ -RELEASE/ -ABORT services.....	386
10.4.8.6 The EventNotification service and protocol	386
10.4.8.7 Transporting long messages.....	386
10.4.8.8 Broadcasting	386
10.4.9 CIASE APDUs.....	386
10.5 The wired and wireless M-Bus profile	389
10.5.1 Scope	389
10.5.2 Targeted communication environments	389
10.5.3 Use of the communication layers for this profile	390
10.5.3.1 Information related to the use of the standard specifying the lower layers	390
10.5.3.2 Structure of the communication profiles	390
10.5.3.3 Lower protocol layers and their use	391
10.5.3.3.1 Physical layer.....	391
10.5.3.3.2 Link layer	391
10.5.3.3.3 Transport layer.....	391
10.5.3.4 Service mapping and adaptation layers	391
10.5.3.4.1 Overview.....	391
10.5.3.4.2 MBUS-DATA service primitives	392
10.5.3.4.2.1 MBUS-DATA.request.....	392
10.5.3.4.2.2 MBUS-DATA.indication	393
10.5.3.4.2.3 MBUS-DATA.confirm.....	393
10.5.3.4.3 MBUS-DATA protocol specification	394
10.5.3.4.3.1 Sending COSEM APDUs	394
10.5.3.4.3.2 Receiving COSEM APDUs	395
10.5.3.5 Registration and connection management	396
10.5.4 Identification and addressing scheme.....	396
10.5.4.1 Overview	396
10.5.4.2 Link Layer Address for wired M-Bus	397
10.5.4.3 Link Layer Address for wireless M-Bus	397
10.5.4.4 Link Layer Address for M-Bus broadcast	398
10.5.4.5 Transport layer address.....	398
10.5.4.6 Application addressing extension – M-Bus wrapper	400
10.5.5 Specific considerations/constraints for using certain services within profile	401
10.5.5.1 Overview	401
10.5.5.2 Application Association establishment and release: ACSE services	401
10.5.5.3 xDLMS services	402
10.5.5.3.1 Request / response type services.....	402
10.5.5.3.2 Unsolicited services	403
10.5.5.3.3 Broadcast messages	403
10.5.5.4 Security mechanisms.....	403
10.5.5.4.1 Transporting long application messages.....	403

10.5.5.5	Media access, bandwidth and timing considerations	403
10.5.6	Communication configuration and management	404
10.5.7	M-Bus frame structures, addressing schemes and examples.....	404
10.5.7.1	General	404
10.5.7.2	None, Short or Long M-Bus Data Header.....	405
10.5.7.2.1	Wired M-Bus	405
10.5.7.3	Wireless M-Bus	406
10.5.7.3.1	Extended Link Layer.....	406
10.5.7.3.2	Transport Layer.....	409
10.5.7.4	Encoding example: Data-Notification carrying daily billing data.	409
10.5.7.4.1	Overview.....	409
10.5.7.4.2	Example: Daily billing data	409
10.5.8	Message sequence charts.....	410
10.6	SMS short wrapper.....	413
10.7	Gateway protocol	414
10.7.1	General.....	414
10.7.2	The gateway protocol.....	415
10.7.3	HES in the WAN/NN acting as Initiator (Pull operation)	416
10.7.4	End devices in the LAN acting as Initiators (Push operation).....	417
10.7.4.1	General	417
10.7.4.2	End device with WAN/NN knowledge.....	417
10.7.4.3	End devices without WAN/NN knowledge	417
10.7.5	Security	417
11	AARQ and AARE encoding examples.....	418
11.1	General	418
11.2	Encoding of the xDLMS InitiateRequest / InitiateResponse APDU	418
11.3	Specification of the AARQ and AARE APDU.....	421
11.4	Data for the examples	422
11.5	Encoding of the AARQ APDU	423
11.6	Encoding of the AARE APDU	426
12	Encoding examples: AARQ and AARE APDUs using a ciphered application context ...	432
12.1	A-XDR encoding of the xDLMS InitiateRequest APDU, carrying a dedicated key	432
12.2	Authenticated encryption of the xDLMS InitiateRequest APDU	432
12.3	The AARQ APDU	433
12.4	A-XDR encoding of the xDLMS InitiateResponse APDU	435
12.5	Authenticated encryption of the xDLMS InitiateResponse APDU.....	436
12.6	The AARE APDU.....	437
12.7	The RLRQ APDU (carrying a ciphered xDLMS InitiateRequest APDU)	438
12.8	The RLRE APDU (carrying a ciphered xDLMS InitiateResponse APDU)	439
13	S-FSK PLC encoding examples	440
13.1	CI-PDUs, ACSE APDUs and xDLMS APDUs carried by MAC frames using the IEC 61334-4-32 LLC sublayer	440
13.2	CI-PDUs, ACSE APDUs and xDLMS APDUs carried by MAC frames using the HDLC based LLC sublayer	445
13.3	Clear Alarm examples	450
14	Data transfer service examples	451
14.1	GET / Read, SET / Write examples.....	451
14.2	ACCESS service example	464

14.3	Compact array encoding example.....	466
14.3.1	General.....	466
14.3.2	The specification of compact-array.....	466
14.3.3	Example 1: Compact array encoding an array of five long-unsigned values.....	467
14.3.4	Example 2: Compact-array encoding of five octet-string values	468
14.3.5	Example 3: Encoding of the buffer of a Profile generic object.....	468
Annex A (normative)	NSA Suite B elliptic curves and domain parameters.....	471
Annex B (informative)	Example of an End entity signature certificate using P-256 signed with P-256.....	473
Annex C (normative)	Use of key agreement schemes in DLMS/COSEM	475
C.1	Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme.....	475
C.2	One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme	478
C.3	Static Unified Model C(0e, 2s, ECC CDH) scheme	482
Annex D (informative)	Exchanging protected xDLMS APDUs between TP and server.....	484
D.1	General	484
D.2	Example 1: Protection is the same in the two directions	484
D.3	Example 2: Protection is different in the two directions	485
Bibliography		487
Index.....		491

Figure 1 – The three steps approach of COSEM: Modelling – Messaging – Transporting	31
Figure 2 – Client–server model and communication protocols	54
Figure 3 – Naming and addressing in DLMS/COSEM	55
Figure 4 – A complete communication session in the CO environment	57
Figure 5 – DLMS/COSEM messaging patterns	59
Figure 6 – DLMS/COSEM generic communication profile	61
Figure 7 – Model of a DLMS/COSEM metering system	62
Figure 8 – DLMS/COSEM server model.....	63
Figure 9 – Model of a DLMS/COSEM client using multiple protocol stacks	64
Figure 10 – Typical PSTN configuration	66
Figure 11 – The location of the physical layer.....	67
Figure 12 – Protocol layer services of the COSEM 3-layer connection-oriented profile	68
Figure 13 – MSC for physical connection establishment.....	72
Figure 14 – MSC for IDENTIFY.request / .response message exchange.....	74
Figure 15 – Handling the Identification service at the server side	74
Figure 16 – Partial state machine for the client side physical layer	75
Figure 17 – MSC for physical connection request.....	77
Figure 18 – Physical connection establishment at the CALLING station.....	78
Figure 19 – MSC for physical connection establishment	79
Figure 20 – Data exchange between the calling and called stations	79
Figure 21 – MSC for a physical disconnection	80
Figure 22 – Entering protocol mode E (HDLC).....	81
Figure 23 – Flow chart and switchover to METERING HDLC in protocol mode E	82

Figure 24 – Physical layer primitives	83
Figure 25 – Physical layer primitives, simplified example with one mode change only	83
Figure 26 – DLMS/COSEM as a standard Internet application protocol	85
Figure 27 – Transport layers of the DLMS/COSEM_on_IP profile	86
Figure 28 – Services of the DLMS/COSEM connection-less, UDP-based transport layer	88
Figure 29 – The wrapper protocol data unit (WPDU).....	90
Figure 30 – The DLMS/COSEM connection-less, UDP-based transport layer PDU (UDP-PDU)	91
Figure 31 – Services of the DLMS/COSEM connection-oriented, TCP-based transport layer	93
Figure 32 – The TCP packet format.....	100
Figure 33 – TCP connection establishment.....	101
Figure 34 – TCP disconnection.....	102
Figure 35 – Data transfer using the COSEM TCP-based transport layer	103
Figure 36 – High-level state transition diagram for the wrapper sublayer	104
Figure 37 – TCP connection state diagram	105
Figure 38 – MSC and state transitions for establishing a transport layer and TCP connection	105
Figure 39 – MSC and state transitions for closing a transport layer and TCP connection	106
Figure 40 – Polling the TCP sublayer for TCP abort indication	107
Figure 41 – Sending an APDU in three TCP packets	108
Figure 42 – Receiving the message in several packets.....	109
Figure 43 – Data link layer services for data link connection	113
Figure 44 – Data link layer services for data link disconnection	117
Figure 45 – Data link layer data transfer services.....	121
Figure 46 – Physical layer services used by the MAC sublayer.....	124
Figure 47 – The ISO/IEC 8802-2 LLC PDU format.....	124
Figure 48 – LLC format as used in DLMS/COSEM.....	124
Figure 49 – MAC sublayer frame format (HDLC frame format type 3)	126
Figure 50 – Multiple frames	126
Figure 51 – The frame format field	126
Figure 52 – Valid server address structures	128
Figure 53 – Address example.....	129
Figure 54 – MSC for long MSDU transfer in a transparent manner	139
Figure 55 – Example configuration to illustrate broadcasting	140
Figure 56 – Sending out a pending UI frame with a .response data	141
Figure 57 – Sending out a pending UI frame with a response to a RR frame	142
Figure 58 – Sending out a pending UI frame on receipt of an empty UI frame.....	143
Figure 59 – State transition diagram for the server MAC sublayer	145
Figure 60 – Layer management services	149
Figure 61 – The structure of the DLMS/COSEM application layers	152
Figure 62 – The concept of composable xDLMS messages	158
Figure 63 – Summary of DLMS/COSEM AL services	161
Figure 64 – Authentication mechanisms	163
Figure 65 – Client – server message security concept.....	166

Figure 66 – End-to-end message security concept	167
Figure 67 – Hash function	169
Figure 68 – Encryption and decryption	170
Figure 69 – Message Authentication Codes (MACs)	171
Figure 70 – GCM functions	172
Figure 71 – Digital signatures	178
Figure 72 – C(2e, 0s) scheme: each party contributes only an ephemeral key pair	179
Figure 73 – C(1e, 1s) schemes: party U contributes an ephemeral key pair, and party V contributes a static key pair	180
Figure 74 – C(0e, 2s) scheme: each party contributes only a static key pair	182
Figure 75 – Architecture of a Public Key Infrastructure (example)	192
Figure 76 – MSC for provisioning the server with CA certificates	201
Figure 77 – MSC for security personalisation of the server	202
Figure 78 – Provisioning the server with the certificate of the client	203
Figure 79 – Provisioning the client / third party with a certificate of the server	204
Figure 80 – Remove certificate from the server	204
Figure 81 – Cryptographic protection of information using AES-GCM	208
Figure 82 – Structure of service-specific global / dedicated ciphering xDLMS APDUs	210
Figure 83 – Structure of general-glo-ciphering and general-ded-ciphering xDLMS APDUs ..	211
Figure 84 – Structure of general-ciphering xDLMS APDUs	212
Figure 85 – Structure of general-signing APDUs	217
Figure 86 – Service primitives	221
Figure 87 – Time sequence diagrams	222
Figure 88 – Additional service parameters to control cryptographic protection and GBT ..	231
Figure 89 – Partial state machine for the client side control function	263
Figure 90 – Partial state machine for the server side control function	264
Figure 91 – MSC for successful AA establishment preceded by a successful lower layer connection establishment	272
Figure 92 – Graceful AA release using the A-RELEASE service	276
Figure 93 – Graceful AA release by disconnecting the supporting layer	277
Figure 94 – Aborting an AA following a PH-ABORT.indication	278
Figure 95 – MSC of the GET service	281
Figure 96 – MSC of the GET service with block transfer	281
Figure 97 – MSC of the GET service with block transfer, long GET aborted	283
Figure 98 – MSC of the SET service	284
Figure 99 – MSC of the SET service with block transfer	285
Figure 100 – MSC of the ACTION service	287
Figure 101 – MSC of the ACTION service with block transfer	288
Figure 102 – Access Service with long response	289
Figure 103 – Access Service with long request and response	289
Figure 104 – MSC of the Read service used for reading an attribute	292
Figure 105 – MSC of the Read service used for invoking a method	293
Figure 106 – MSC of the Read Service used for reading an attribute, with block transfer ...	293

Figure 107 – MSC of the Write service used for writing an attribute	296
Figure 108 – MSC of the Write service used for invoking a method	297
Figure 109 – MSC of the Write Service used for writing an attribute, with block transfer	297
Figure 110 – MSC of the Unconfirmed Write service used for writing an attribute	298
Figure 111 – Partial service invocations and GBT APDUs	301
Figure 112 – GET service with GBT, switching to streaming	303
Figure 113 – GET service with partial invocations, GBT and streaming, recovery of 4 th block sent in the 2nd stream	304
Figure 114 – GET service with partial invocations, GBT and streaming, recovery of 4 th and 5 th block	305
Figure 115 – GET service with partial invocations, GBT and streaming, recovery of last block	306
Figure 116 – SET service with GBT, with server not supporting streaming, recovery of 3rd block	307
Figure 117 – ACTION-WITH-LIST service with bi-directional GBT and block recovery	308
Figure 118 – DataNotification service with GBT with partial invocation	309
Figure 119 – Identification/addressing scheme in the 3-layer, CO, HDLC based communication profile	348
Figure 120 – Summary of data link layer services	349
Figure 121 – Example: EventNotification triggered by the client	352
Figure 122 – Multi-drop configuration and its model	353
Figure 123 – Master/ Slave operation on the multi-drop bus	353
Figure 124 – Communication architecture	355
Figure 125 – Examples for lower-layer protocols in the TCP-UDP/IP based profile(s)	356
Figure 126 – Identification / addressing scheme in the TCP-UDP/IP based profile(s)	358
Figure 127 – Summary of TCP / UDP layer services	359
Figure 128 – Communication architecture	363
Figure 129 – The DLMS/COSEM S-FSK PLC communication profile	364
Figure 130 – Co-existence of the connectionless and the HDLC based LLC sublayers	366
Figure 131 – Intelligent Search Initiator process flow chart	374
Figure 132 – The Discovery and Registration process	376
Figure 133 – MSC for the discovery and registration process	381
Figure 134 – MSC for successful confirmed AA establishment	382
Figure 135 – MSC for releasing an Application Association	383
Figure 136 – MSC for an EventNotification service	384
Figure 137 – MSC for the Discovery and Registration process	385
Figure 138 – MSC for successful confirmed AA establishment and the GET service	385
Figure 139 – Entities and interfaces of a smart metering system using the terminology of IEC 62056-1-0	389
Figure 140 – The DLMS/COSEM wired and wireless M-Bus communication profiles	390
Figure 141 – Summary of DLMS/COSEM M-Bus based TL services	392
Figure 142 – Identification and addressing scheme in the wired M-Bus profile	397
Figure 143 – Link Layer Address for wireless M-Bus	398
Figure 144 – M-Bus TPDU formats	399
Figure 145 – CI _{TL} without M-Bus Data Header	399

Figure 146 – M-Bus communication paths direct or cascaded	404
Figure 147 – Wired M-Bus frame structure, none M-Bus Data Header	406
Figure 148 – Wired M-Bus frame structure with long M-Bus Data Header.....	406
Figure 149 – Wireless M-Bus frame structure with short ELL, none M-Bus Data Header....	407
Figure 150 – Wireless M-Bus frame structure with long ELL, none M-Bus Data Header.....	408
Figure 151 – Wireless M-Bus frame structure with long ELL and long M-Bus Data Header .	408
Figure 152 – Daily billing data without / with DLMS/COSEM security applied	410
Figure 153 – MSC for the COSEM-OPEN service for wired M-Bus, none M-Bus header	411
Figure 154 – MSC the GET service for wired M-Bus, none M-Bus header	412
Figure 155 – Short wrapper	413
Figure 156 – General architecture with gateway	414
Figure 157 – The fields used for pre-fixing the COSEM APDUs	415
Figure 158 – Pull message sequence chart	416
Figure 159 – Push message sequence chart	417
Figure C. 1 – MSC for key agreement using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme.	475
Figure C. 2 – Ciphered xDLMS APDU protected by an ephemeral key established using the One-pass Diffie-Hellman (1e, 1s, ECC CDH) scheme	478
Figure C. 3 – Ciphered xDLMS APDU protected by an ephemeral key established using the Static Unified Model C(0e, 2s, ECC CDH) scheme	482
Figure D. 1 – Exchanging protected xDLMS APDUs between TP and server: example 1	485
Figure D. 2 – Exchanging protected xDLMS APDUs between TP and server: example 2	486
Table 1– Client and server SAPs	56
Table 2 – Reserved wrapper port numbers in the UDP-based DLMS/COSEM TL.....	91
Table 3 – State transition table of the client side LLC sublayer.....	125
Table 4 – State transition table of the server side LLC sublayer	125
Table 5 – Table of reserved client addresses	128
Table 6 – Table of reserved server addresses	128
Table 7 – Handling inopportune address lengths	130
Table 8 – Control field bit assignments of command and response frames	130
Table 9 – Example for parameter negotiation values with the SNRM/UA frames	136
Table 10 – Summary of MAC addresses for the example.....	140
Table 11 – Broadcast UI frame handling.....	141
Table 12 – Clarification of the meaning of PDU size for DLMS/COSEM	160
Table 13 – Elliptic curves in DLMS/COSEM security suites	176
Table 14 – Ephemeral Unified Model key agreement scheme summary	180
Table 15 – One-pass Diffie-Hellman key agreement scheme summary	181
Table 16 – Static Unified Model key agreement scheme summary	182
Table 17 – OtherInfo subfields and substrings	184
Table 18 – Security algorithm ID-s	184
Table 19 – DLMS/COSEM security suites	185
Table 20 – Symmetric keys types	187

Table 21 – Key information with general-ciphering APDU and data protection	188
Table 22 – Asymmetric keys types and their use	190
Table 23 – X.509 v3 Certificate structure.....	194
Table 24 – X.509 v3 tbsCertificate fields	194
Table 25 – Naming scheme for the Root-CA instance (informative)	195
Table 26 – Naming scheme for the Sub-CA instance (informative)	195
Table 27 – Naming scheme for the end entity instance	196
Table 28 – X.509 v3 Certificate extensions.....	197
Table 29 – Key Usage extensions	198
Table 30 – Subject Alternative Name values	199
Table 31 – Issuer Alternative Name values.....	199
Table 32 – Basic constraints extension values	199
Table 33 – Certificates handled by DLMS/COSEM end entities	200
Table 34 – Security policy values (“Security setup” version 1)	205
Table 35 – Access rights values (“Association LN” ver 3 “Association SN” ver 4)	206
Table 36 – Ciphered xDLMS APDUs	207
Table 37 – Security control byte	209
Table 38 – Plaintext and Additional Authenticated Data.....	209
Table 39 – Use of the fields of the ciphering xDLMS APDUS	213
Table 40 – Example: glo-get-request xDLMS APDU	213
Table 41 – ACCESS service with general-ciphering, One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) key agreement scheme.....	215
Table 42 – DLMS/COSEM HLS authentication mechanisms	218
Table 43 – HLS example using authentication-mechanism5 with GMAC	219
Table 44 – HLS example using authentication-mechanism 7 with ECDSA	220
Table 45 – Codes for AL service parameters	223
Table 46 – Service parameters of the COSEM-OPEN service primitives	224
Table 47 – Service parameters of the COSEM-RELEASE service primitives.....	228
Table 48 – Service parameters of the COSEM-ABORT service primitives.....	230
Table 49 – Additional service parameters	232
Table 50 – Security parameters.....	233
Table 51 – APDUs used with security protection types	235
Table 52 – Service parameters of the GET service	236
Table 53 – GET service request and response types	237
Table 54 – Service parameters of the SET service	239
Table 55 – SET service request and response types	240
Table 56 – Service parameters of the ACTION service	242
Table 57 – ACTION service request and response types	243
Table 58 – Service parameters of the ACCESS service	248
Table 59 – Service parameters of the DataNotification service primitives	251
Table 60 – Service parameters of the EventNotification service primitives	252
Table 61 – Service parameters of the TriggerEventNotificationSending.request service primitive	253

Table 62 – Variable Access Specification	253
Table 63 – Service parameters of the Read service.....	254
Table 64 – Use of the Variable_Access_Specification variants and the Read.response choices	255
Table 65 – Service parameters of the Write service.....	257
Table 66 – Use of the Variable_Access_Specification variants and the Write.response choices	258
Table 67 – Service parameters of the UnconfirmedWrite service	260
Table 68 – Use of the Variable_Access_Specification variants	260
Table 69 – Service parameters of the InformationReport service	261
Table 70 – Service parameters of the SetMapperTable.request service primitives	261
Table 71 – Summary of ACSE services	262
Table 72 – Summary of xDLMS services	262
Table 73 – Functional Unit APDUs and their fields	266
Table 74 – COSEM application context names	269
Table 75 – COSEM authentication mechanism names	270
Table 76 – Cryptographic algorithm ID-s	270
Table 77 – xDLMS Conformance block.....	279
Table 78 – GET service types and APDUs	280
Table 79 – SET service types and APDUs	284
Table 80 – ACTION service types and APDUs.....	287
Table 81 – Mapping between the GET and the Read service	291
Table 82 – Mapping between the ACTION and the Read service	291
Table 83 – Mapping between the SET and the Write service	294
Table 84 – Mapping between the ACTION and the Write service	295
Table 85 – Mapping between the SET and the UnconfirmedWrite service.....	298
Table 86 – Mapping between the ACTION and the UnconfirmedWrite service	298
Table 87 – Mapping between the EventNotification and InformationReport services	299
Table 88 – Application associations and data exchange in the 3-layer, CO, HDLC based profile.....	350
Table 89 – Application associations and data exchange in the TCP-UDP/IP based profile ..	360
Table 90 – Service parameters of the Discover service primitives	367
Table 91 – Service parameters of the Register service primitives	368
Table 92 – Service parameters of the PING service primitives.....	368
Table 93 – Service parameters of the RepeaterCall service primitives.....	369
Table 94 – Service parameters of the ClearAlarm service primitives.....	372
Table 95 – MAC addresses	379
Table 96 – Reserved IEC 61334-4-32 LLC addresses on the client side	380
Table 97 – Reserved IEC 61334-4-32 LLC addresses on the server side.....	380
Table 98 – Reserved HDLC based LLC addresses on the client side	380
Table 99 – Reserved HDLC based LLC addresses on the server side	380
Table 100 – Source and Destination APs and addresses of CI-PDUs	380
Table 101 – Application associations and data exchange in the S-FSK PLC profile using the connectionless LLC sublayer.....	383
Table 102 – Wired M-Bus Link Layer Addresses	397

Table 103 – DLMS/COSEM M-Bus based TL CI _{TL} values	399
Table 104 – CI fields used for link management purposes	400
Table 105 – Client and server SAPs	401
Table 106 – Application associations and data exchange in the M-Bus based profiles	402
Table 107 – Example: Daily billing data.....	409
Table 108 – Reserved Application Process SAPs.....	413
Table 109 – Conformance block	419
Table 110 – A-XDR encoding the xDLMS InitiateRequest APDU	420
Table 111 – A-XDR encoding the xDLMS InitiateResponse APDU.....	421
Table 112 – BER encoding the AARQ APDU.....	424
Table 113 – The complete AARQ APDU	426
Table 114 – BER encoding the AARE APDU	427
Table 115 – The complete AARE APDU	431
Table 116 – A-XDR encoding of the xDLMS InitiateRequest APDU	432
Table 117 – Authenticated encryption of the xDLMS InitiateRequest APDU.....	433
Table 118 – BER encoding of the AARQ APDU	434
Table 119 – A-XDR encoding of the xDLMS InitiateResponse APDU.....	435
Table 120 – Authenticated encryption of the xDLMS InitiateResponse APDU	436
Table 121 – BER encoding of the AARE APDU	437
Table 122 – BER encoding of the RLRQ APDU	438
Table 123 – BER encoding of the RLRE APDU	439
Table 124 – The objects used in the examples	451
Table 125 – Example: Reading the value of a single attribute without block transfer	452
Table 126 – Example: Reading the value of a list of attributes without block transfer.....	452
Table 127 – Example: Reading the value of a single attribute with block transfer	453
Table 128 – Example: Reading the value of a list of attributes with block transfer	455
Table 129 – Example: Writing the value of a single attribute without block transfer	457
Table 130 – Example: Writing the value of a list of attributes without block transfer	458
Table 131 – Example: Writing the value of a single attribute with block transfer	459
Table 132 – Example: Writing the value of a list of attributes with block transfer	460
Table 133 – Example: ACCESS service without block transfer	464
Table A. 1 – ECC_P256_Domain_Parameters.....	471
Table A. 2 – ECC_P384_Domain_Parameters.....	472
Table C. 1 – Test vector for key agreement using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme.....	476
Table C. 2 – Test vector for key agreement using the One-pass Diffie-Hellman (1e, 1s, ECC CDH) scheme.....	479

Foreword

Copyright

© Copyright 1997-2015 DLMS User Association.

This Edition 8.1 of the Green Book integrates Corrigendum 1 to Edition 8.0:2014.

For Technical corrigenda see the Table below. The Editorial changes can be found in Green Book Edition 8.0 Corrigendum 1.

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 166 countries worldwide and other treaties apply.

Acknowledgement

The actual 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 standardization

The contents of this edition will be used to prepare a revision of IEC 62056-5-3:—, Electricity metering data exchange – The DLMS/COSEM suite – Part 5-3: DLMS/COSEM application layer.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	25/500
-----------------------	------------	------------------------	--------

Revision history

Version	Date	Author	Comment
Release 1	1 st April 1998	DLMS UA	Initial version
First Edition	1 st May 2000	DLMS UA	Major rework, adapted to CDVs of IEC TC13
Second Edition	15 th May 2001	DLMS UA	Considering comments to CDVs by IEC National Committees
Third Edition	30 th March 2002	DLMS UA	Content adapted to IEC International Standards
Fourth Edition	15 th 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 th August 2005	DLMS-UA	Content aligned with IEC TC 13 CDV-s and comments received.
Sixth Edition	27 th 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 nd December 2009	DLMS UA	Includes: <ul style="list-style-type: none">- SN block transfer- Data security- S-FSK PLC profile
Eighth Edition	4 th July 2014	DLMS UA	Includes (See also the detailed list): <ul style="list-style-type: none">- Security extensions;- Compression;- ACCESS service;- DataNotification service;- General block transfer mechanism;- XML schema;- Wired and wireless M-Bus profile;- SMS profile;- Gateway protocol;- Compact array encoding example.
Eighth Edition corrected	7 th July 2014	DLMS UA	Wrong Figure 139 replaced with the correct one. Missing CIASE APDU module added as 10.4.9. Filename: Green Book 8.1_Released_151214
Edition 8.0 Corrigendum 1	14 th December	DLMS UA	Technical and editorial Corrigendum 1 to Edition 8.0.
Edition 8.1	14th th December	DLMS UA	Consolidated edition integrating Corrigendum 1.

Main technical changes in this Technical Report compared to Green Book 7 + Amendment 3:

Item	Clause	<i>Highlights in the text indicate Clause Figures and Tables where changes have been made.</i>
1.	1	Scope: Text on security added.
2.	2	New references added.
3.	3.1	General DLMS/COSEM definitions added
4.	3.2	Definitions related to cryptographic security added
5.	3.4	General abbreviations: new abbreviations added
6.	3.3	Definitions, abbreviations, symbols and notation relevant for the Galois/Counter Mode brought here from Green Book Ed.7 9.2.4.8.2
7.	3.6	Definitions, abbreviations, symbols and notation relevant for the ECDSA algorithm added.
8.	3.7	Definitions, abbreviations, symbols and notation relevant for the key agreement algorithms added
9.	3.8	Abbreviations related to the DLMS/COSEM M-Bus communication profile added.
10.	4	Specifies the general concepts of information exchange in DLMS/COSEM, extending Green Book Ed. 7 Clause 4.
11.	4.1	General: Text updated, key characteristics of DLMS/COSEM listed here.
12.	4.2	Communication model: new text introducing APs, AEs, ASEs, AAs and their relationship. Client/server model is explained here.
13.	4.3	Naming and addressing: new text bringing all related elements together, and bringing in the concept of data exchange with third parties. Table of SAPs added.
14.	4.3.4	System title: Green Book Ed. 7 9.2.4.8.3.4.6 on exchanging system titles brought here.
15.	4.3.6	Client user identification added.
16.	4.4	Connection oriented operation: Text of Green Book 7 4.2 included here in a revised form.
17.	4.5	Application associations. New text, bringing all related elements together.
18.	4.6	Messaging patterns: new text bringing in the concept of push operation.
19.	4.7	Data exchange between third parties and DLMS/COSEM servers: new text.
20.	4.8	Communication profiles: Green book Edition 7 subclause 4.1 brought here in a revised form.
21.	4.9	Application models: Green Book Ed. 7 subclause 4.5 brought here.
22.	4.10	Model of DLMS/COSEM servers: Green Book Ed. 7 subclause 4.6 brought here.
23.	4.11	Model of DLMS/COSEM clients: Green Book Ed. 7 subclause 4.7 brought here.
24.	4.12	Interoperability and interconnectivity in COSEM: Green Book Ed. 7 subclause 4.3 is revised.
25.	4.13	Ensuring interconnectivity: the protocol identification service: Green Book Edition 7 4.4 is reproduced here in a slightly revised form.
26.	7	DLMS/COSEM transport layer for IP networks: Text replaced as in IEC 62056-4-7. It covers now both IP4 and IPv6.
27.	9.1	The whole clause has been revised to present the additional services and mechanisms compared to DLMS as specified in IEC 61334-4-41:1996.
28.	9.2	Information security in DLMS/COSEM: the whole clause has been reworked to describe the extended security features of DLMS/COSEM. Main elements: <ul style="list-style-type: none"> - the DLMS/COSEM security concept is presented; - not only xDLMS APDUs but also COSEM data – carried by the APDUs – can be protected; - the protection can be applied not only between clients and servers but also between third parties and servers via clients; - symmetric and public key algorithms are available to provide any combination of authentication, encryption and digital signature; - multiple layers of protection can be applied and verified by multiple entities; - key-transport has been complemented by key agreement; - compression (managed together with symmetric key algorithms) added.
29.	9.3.2	The COSEM-OPEN service: Service parameters and their use updated to support client user identification and transportation of Certificates.

Item	Clause	<i>Highlights in the text indicate Clause Figures and Tables where changes have been made.</i>
30.	9.3.5	Protection and general block transfer parameters: clause reworked to cover the new general protection and general block transfer mechanisms and related parameters.
31.	9.3.6 9.3.7	GET, SET, ACTION service: either the service-specific or the general block transfer mechanism can be used.
32.	9.3.8	The ACTION service: Text on method invocation parameters – use of null-data – precised.
33.	9.3.9	The ACCESS service: the new unified service added, with the main features presented.
34.	9.3.10	The DataNotification service: The new service added.
35.	9.3.14, 9.3.15	Read, Write service: Either the service-specific or the general block transfer mechanism can be used.
36.	9.3.16	UnconfirmedWrite service: General block transfer mechanism can be used.
37.	9.3.19	Summary of services and LN/SN data transfer service mapping: New services added.
38.	9.4.1.1 9.4.1.2	State definitions: Updated to include the new services and APDUs.
39.	9.4.2.1	ACSE functional units, services and service parameters: Updated to be in line with ISO/IEC 15953:1999 and ISO/IEC 15954:1999 replacing ISO/IEC 8649 and ISO/IEC 8650.
40.	9.4.2.2.3	The COSEM authentication mechanism name: New mechanisms names added.
41.	9.4.2.2.4	Cryptographic algorithm ID-s: Added, these IDs are used in the KDF function.
42.	9.4.3	APDU encoding rules: Encoding of the ACSE APDUs, the xDLMS APDUs and XML.
43.	9.4.4.1	Protocol for the establishment of confirmed application associations: text amended to include parsing of the new parameters.
44.	9.4.6.1	Negotiation of services and options – the conformance block: New elements added.
45.	9.4.6.3 9.4.6.4 9.4.6.5	Protocol of the GET, SET, and ACTION service: It is specified that either the service-specific or the general block transfer mechanism can be used.
46.	9.4.6.6	Protocol for the ACCESS service added
47.	9.4.6.7	Protocol of the DataNotification service added
48.	9.4.6.9 9.4.6.10	Protocol of the Read and Write service: It is specified that either the service-specific or the general block transfer mechanism can be used.
49.	9.4.6.13	Protocol of general block transfer mechanism added
50.	9.5	Abstract syntax of COSEM APDUs: abstract syntax of the new services and APDUs added.
51.	9.6	COSEM APDU XML schema added
52.	10.2.6.1	Confirmed and unconfirmed AAs and data transfer service invocations, frame types used: it is allowed now that HDLC I frames carry both confirmed and unconfirmed service invocations.
53.	10.3	The TCP-UDP/IP based communication profiles (COSEM_on_IP): Text amended to be in line with IEC 62056-9-7, supporting both IPv4 and IPv6.
54.	10.4	The S-FSK PLC profile: Text amended to be in line with IEC 62056-9-7. No technical changes have been made.
55.	10.5	The wired and wireless M-Bus profil added.
56.	10.6	SMS short wrapper added.
57.	10.7	Gateway protocol added.
58.	14.2	Data transfer service examples: Example for the ACCESS service added.
59.	14.3	Compact array example added.
60.	Annex A	NSA Suite B elliptic curves and domain parameters added
61.	Annex B	Certificate examples added.
62.	Annex C	Use of key agreement schemes in DLMS/COSEM added
63.	Annex D	Exchanging protected xDLMS APDUs between TP and server added.

Item	Clause	<i>Highlights in the text indicate Clause Figures and Tables where changes have been made.</i>
64.	Bibliography	New references added.

List of main technical changes in Edition 8.1:

Item	Clause	Corrigendum
1.	Foreword	Entry on Intellectual Property Rights removed; the patent EP 0 638 886 B1 concerning the RepeaterCall service has expired in 2014.
2.	2	Referenced documents ITU-T X.811:1995, <i>Information technology - Open Systems Interconnection - Security frameworks for open systems: Authentication framework</i> added.
3.	9.2.2.2.2.4	<i>Second and the third paragraphs under the bullet points replaced:</i> 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. NOTE The dedicated-key, if transferred, can be used from this moment.
4.	9.2.7.3,	<i>Text of 4th bullet point amended:</i> <i>Existing text:</i> "to apply encryption, authentication or authenticated encryption, the general-ciphering APDU is used. Authenticated encryption is considered to be a single layer of protection;" <i>New text:</i> "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;"
5.	9.2.7.3	<i>Second para below the bullet points:</i> "Both APDUs include..." replaced by "Both the general-ciphering and general-signing APDUs include...."
6.	9.2.7.3	<i>Text of the 3rd para below the bullet points amended: the following has been added in front of the existing text:</i> 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....
7.	9.3.2	<i>9th para below Table 46, second line:</i> "it shall carry" replaced by "it may carry".
8.	9.3.2	5 th para below NOTE 1, Text amended to read: The use of the Responding_AP_Title parameter is conditional. When the Application_Context_Name parameter indicates an application context using ciphering or when the HLS authentication mechanism requires the use of the system title, it shall carry the server system title specified in 4.3.4.
9.	9.3.2	<i>2nd and 3rd bullet point below Table 46:</i> <i>At the end added (twice):</i> and it shall indicate authentication.
10.	9.3.3	In the paragraph starting with "If the xDLMS InitiateRequest APDU can be successfully deciphered" "...authenticated and encrypted the same way as in the AARE..." replaced by "...protected the same way as in the AARE..."

11.	10.2.6.1	<i>NOTE to Table 88 added:</i> NOTE As described above this table, when the meter 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.
12.	10.5	<i>The whole subclause has been replaced with the text of draft IEC 62056-7-3. Main technical changes:</i> <ul style="list-style-type: none">- Restructured to be in line with IEC/TS 62056-1-1, <i>ELECTRICITY METERING DATA EXCHANGE – THE DLMS/COSEM SUITE – Part 1-1: Template for DLMS/COSEM communication profile standards</i>;- The profile covers now also the H1 and H2 interfaces of the smart metering reference architecture;- Examples for frame structures, addressing schemas and encoding have been added;- Example message sequence charts have been added.
13.	C.1	In Figure C.1, The step “generate ephemeral key pair” by Part V was in the wrong position. Figure replaced.
14.	C.1	The result of the KDF function was erroneous. Table C.1 replaced.
15.	C.2	The result of the KDF function is erroneous. Table C.2 replaced.
16.	Bibliography	IEC 60870-5-1:1990, <i>Telecontrol equipment and systems. Part 5: Transmission protocols – Section One: Transmission frame formats</i> added.

1 Scope

The DLMS/COSEM specification specifies an interface model and communication protocols for data exchange with metering equipment.

The interface model provides a view of the functionality of the meter 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 metering equipment 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.

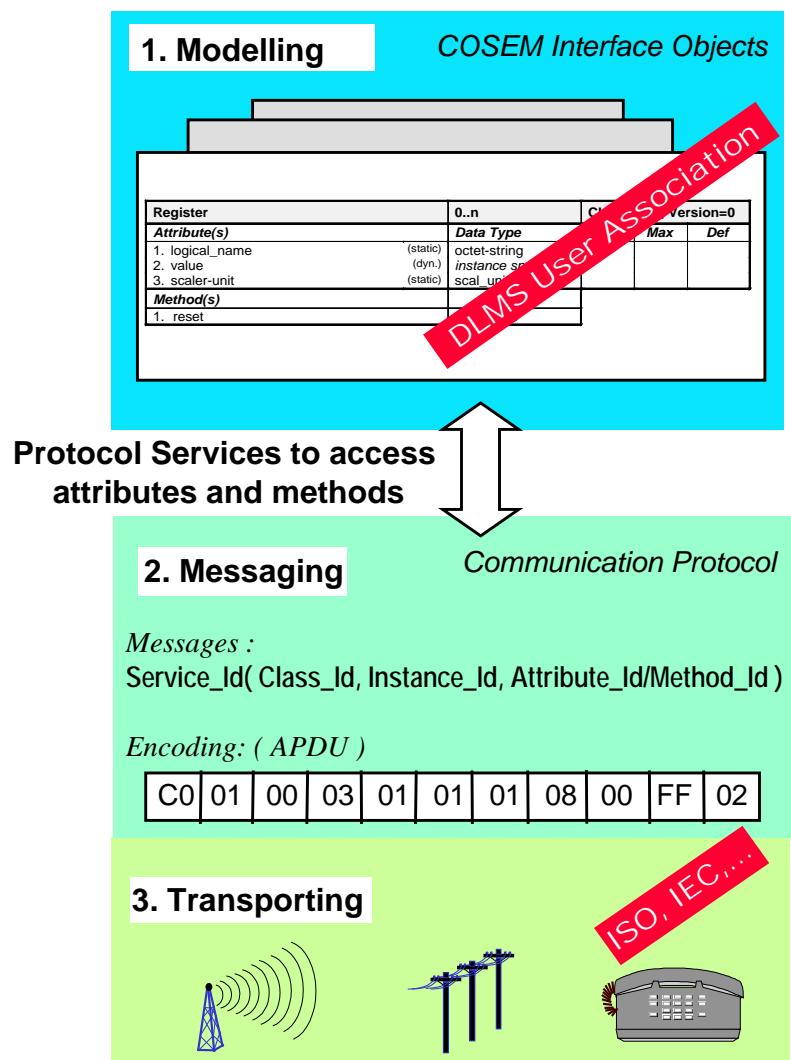


Figure 1 – The three steps approach of COSEM: Modelling – Messaging – Transporting

Step 1 is specified in the document "COSEM interface classes and the OBIS identification system" DLMS UA 1000-1 Ed. 12.1:2015. 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 meter.

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 metering systems requires strong information security mechanisms to protect the privacy of energy consumers, the business interests of the energy and service providers and the security of the energy 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.

Growing privacy and security concerns require – and technology developments enable – further extending the security mechanisms.

This Technical Report specifies such extensions while keeping backwards compatibility. The most important new elements are:

- not only xDLMS APDUs but also COSEM data carried by the APDUs can be protected;
- the protection can be applied not only between clients and servers but also between third parties and servers via clients providing end-to-end security;
- symmetric and public key algorithms are available to provide any combination of authentication, encryption and digital signature;
- multiple layers of protection can be applied and verified by multiple entities;
- key-transport has been complemented by key agreement.

NOTE 1 COSEM data include attribute values as well as method invocation and return parameters.

NOTE 2 Third parties are parties other than DLMS/COSEM clients and servers, and may be for example market participants' ERP systems.

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".

2 Referenced documents (see also the Bibliography)

Ref.	Title
DLMS UA 1000-1 Ed. 12.1:2015	<i>COSEM Interface Classes and OBIS Identification System, the "Blue Book"</i>
DLMS UA 1000-1	<i>COSEM Interface Classes and OBIS Identification System, the "Blue Book"</i> NOTE This undated reference is used unless a specific clause needs to be referenced.
DLMS UA 1001-1	<i>DLMS/COSEM Conformance test and certification process, the "Yellow Book"</i>
DLMS UA 1002 Ed. 1.0:2003	<i>COSEM Glossary of Terms, "White Book"</i>
IEC 61334-4-1:1996	<i>Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 1: Reference model of the communication system</i>
IEC 61334-4-32:1996	<i>Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 32: Data link layer – Logical link control (LLC)</i>
IEC 61334-4-41:1996	<i>Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 41: Application protocol – Distribution line message specification</i>
IEC 61334-4-511:2000	<i>Distribution automation using distribution line carrier systems – Part 4-511: Data communication protocols – Systems management – CIASE protocol</i>
IEC 61334-4-512:2001	<i>Distribution automation using distribution line carrier systems – Part 4-512: Data communication protocols – System management using profile 61334-5-1 – Management Information Base (MIB)</i>
IEC 61334-5-1:2001	<i>Distribution automation using distribution line carrier systems – Part 5-1: Lower layer profiles – The spread frequency shift keying (S-FSK) profile</i>
IEC 61334-6:2000	<i>Distribution automation using distribution line carrier systems – Part 6: A-XDR encoding rule</i>

IEC 62056-1-0	<i>Electricity metering data exchange – The DLMS/COSEM suite – Part 1 0: Smart metering standardisation framework</i>
IEC 62056-21:2002	<i>Electricity metering – Data exchange for meter reading, tariff and load control – Part 21: Direct local data exchange</i>
ISO/IEC 7498-1:1994,	<i>Information technology - Open Systems Interconnection - Basic Reference Model: The Basic Model</i>
ISO/IEC 8649 Ed. 2.0:1996	<i>Information technology – Open Systems Interconnection – Service definition for the Association Control Service Element</i> NOTE This standard has been replaced by ISO/IEC 15953:1999
ISO/IEC 8650-1 Ed 2.0:1996	<i>Information technology – Open systems interconnection – Connection-oriented protocol for the association control service element: Protocol specification</i> NOTE This standard has been replaced by ISO/IEC 15954:1999
ISO/IEC 8802-2 Ed. 3.0:1998	<i>Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 2: Logical link control</i>
ISO/IEC 8824:2008	<i>Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation</i>
ISO/IEC 8825:2008	<i>Information technology - ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)</i>
ISO/IEC 9798-1:2010	<i>Information technology — Security techniques — Entity authentication — Part 1: General</i>
ISO/IEC 13239:2002	<i>Information Technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures</i>
ISO/IEC 15953:1999	<i>Information technology — Open Systems Interconnection — Service definition for the Application Service Object Association Control Service Element</i> 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	<i>Information technology — Open Systems Interconnection — Connection-mode protocol for the Application Service Object Association Control Service Element</i> 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.
ITU-T V.44: 2000	<i>SERIES V: DATA COMMUNICATION OVER THE TELEPHONE NETWORK – Error control – V.44:2000, Data compression procedures</i>
ITU-T X.509:2008	<i>SERIES X: DATA NETWORKS, OPEN SYSTEM COMMUNICATIONS AND SECURITY – Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks</i>
ITU-T X.693 (11/2008)	<i>Information technology – ASN.1 encoding rules: XML Encoding Rules (XER)</i>
ITU-T X.693 Corrigendum 1(10/2011)	<i>Information technology – ASN.1 encoding rules: XML Encoding Rules (XER) Technical Corrigendum 1</i>
ITU-T X.694 (11/2008)	<i>Information technology – ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1</i>
ITU-T X.694 Corrigendum 1 (10/2011)	<i>Information technology – ASN.1 encoding rules: Mapping W3C XML schema definitions into ASN.1Technical Corrigendum 1</i>
ITU-T X.811:1995	<i>Information technology - Open Systems Interconnection - Security frameworks for open systems: Authentication framework</i>
CEN/CLC/ETSI TR 50572	<i>Functional reference architecture for communications in smart metering systems</i>
EN13757-1:2014	<i>Communication system for and remote reading of meters – Part 1: Data exchange</i>
EN 13757-2:2004	<i>Communication system for and remote reading of meters – Part 2 : Physical and Link Layer, Twisted Pair Baseband (M-Bus)</i>
EN13757-3:2013	<i>Communication system for and remote reading of meters – Part 3: Dedicated application layer</i>
EN 13757-4:2013	<i>Communication system for and remote reading of meters – Part 4: Wireless meter (Radio meter reading for operation in SRD bands)</i>
EN13757-5:2015	<i>Communication system for and remote reading of meters – Part 5: Wireless relaying</i>
EN 13757-6:2015	<i>Communication system for meters – Part 6: Local Bus</i>

ANSI C12.21:1999	<i>Protocol Specification for Telephone Modem Communication</i>
FIPS PUB 180-4:2012	<i>Secure Hash Standard (SHS)</i>
FIPS PUB 186-4:2013	<i>Digital Signature Standard (DSS)</i>
FIPS PUB 197:2001	<i>Advanced Encryption Standard (AES)</i>
NIST SP 800-21:2005	<i>Guideline for Implementing Cryptography in the Federal Government</i>
NIST SP 800-32:2001	<i>Introduction to Public Key Technology and the Federal PKI Infrastructure</i>
NIST SP 800-38D:2007	<i>Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC</i>
NIST SP 800-56A Rev. 2: 2013	<i>Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography</i>
NIST SP 800-57:2012	<i>Recommendation for Key Management – Part 1: General (Revision 3)</i>
NSA1	<i>Suite B Implementer's Guide to FIPS 186-3 (ECDSA), Feb 3rd 2010</i>
NSA2	<i>Suite B Implementer's Guide to NIST SP800-56A, 28th July 2009</i>
NSA3	<i>NSA Suite B Base Certificate and CRL Profile, 27th May 2008</i>
RFC 3394	<i>Advanced Encryption Standard (AES) Key Wrap Algorithm, 2002, http://tools.ietf.org/html/rfc3394</i>
RFC 4108	<i>Using Cryptographic Message Syntax (CMS) to Protect Firmware Packages, 2005, http://www.ietf.org/rfc/rfc4108</i>
RFC 4210	<i>Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP), 2005, http://www.ietf.org/rfc/rfc4210.txt</i>
RFC 5280	<i>Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008, http://www.ietf.org/rfc/rfc5280</i>
RFC 5349	<i>Elliptic Curve Cryptography (ECC) Support for Public Key Cryptography for Initial Authentication in Kerberos (PKINIT), 2008, https://tools.ietf.org/html/rfc5349</i>
STD0005 (1981)	<i>Internet Protocol. Also: RFC0791, RFC0792, RFC0919, RFC0922, RFC0950, RFC1112</i>
STD0006 (1980)	<i>User Datagram Protocol. Also: RFC0768</i>
STD0007 (1981)	<i>Transmission Control Protocol. Also: RFC0793</i>

3 Terms, Definitions and Abbreviations

3.1 General DLMS/COSEM definitions

3.1.1

ACSE APDU

an APDU used by the Association Control Service Element (ACSE)

3.1.2

application association

a 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 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

client

an application process running in the data collection system

3.1.8

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

3.1.9

COSEM

Companion Specification for Energy Metering; refers to the COSEM object model

3.1.10

COSEM APDU

comprises ACSE APDUs and xDLMS APDUs

3.1.11

COSEM data

COSEM object attribute values, method invocation and return parameters

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	35/500
-----------------------	------------	------------------------	--------

3.1.12**COSEM interface class**

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

3.1.13**COSEM object**

an instance of a COSEM interface class

3.1.14**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.

3.1.15**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]

3.1.16**entity authentication**

corroboration that an entity is the one claimed

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

3.1.17**logical device**

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

3.1.18**master**

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

3.1.19**mutual authentication**

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

Note 1 to entry: The DLMS/COSEM HLS authentication mechanism provides mutual authentication.

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

3.1.20**physical device**

a physical metering equipment, the highest level element used in the COSEM interface model of metering equipment

3.1.21**pull operation**

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

3.1.22**push operation**

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

3.1.23**system title**

a unique identifier of the system

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	36/500
-----------------------	------------	------------------------	--------

3.1.24**server**

an application process running in a metering equipment

3.1.25 slave

station responding to requests of a master station

Note 1 to entry: A meter is normally a slave station.

3.1.26**unilateral authentication**

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

Note 1 to entry: The DLMS/COSEM LLS authentication mechanism provides unilateral authentication.

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

3.1.27**xDLMS**

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

3.1.28**xDLMS APDU**

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

3.1.29**xDLMS message**

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

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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	37/500
-----------------------	------------	------------------------	--------

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 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 1 to entry:

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	38/500
-----------------------	------------	------------------------	--------

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]

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 1 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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	39/500
-----------------------	------------	------------------------	--------

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]

3.2.30**key**

see cryptographic key

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	40/500
-----------------------	------------	------------------------	--------

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.

[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 1 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	41/500
-----------------------	------------	------------------------	--------

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

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]

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	2015-12-14	DLMS UA 1000-2 Ed. 8.1	42/500
-----------------------	------------	------------------------	--------

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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	43/500
-----------------------	------------	------------------------	--------

[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]

3.2.62

session key

a cryptographic key established for use for a relatively short period of time.

Note 1 to entry: In DLMS/COSEM the dedicated key is a session key.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	44/500
-----------------------	------------	------------------------	--------

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**

the data upon which a digital signature has been computed

3.2.67**static symmetric 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

Note 1 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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	45/500
-----------------------	------------	------------------------	--------

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 3.4.1 to 3.4.13 abbreviations and symbols in this subclause is NIST SP 800-38D:2007.

3.3.1

Additional Authenticated Data, AAD

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

3.3.2

authenticated decryption

the 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

the 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)

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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	46/500
-----------------------	------------	------------------------	--------

3.3.5**block cipher**

a 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**

the 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**

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

Note 1 to entry: For the purposes of this standard, the invocation field is the invocation counter.

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**

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

3.3.13**plaintext, P**

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

3.3.14**security control byte, SC**

a 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* || *IC*.

3.4 General abbreviations

Abbreviation	Meaning
.cnf	.confirm service primitive
.ind	.indication service primitive
.req	.request service primitive
.res	.response service primitive
AA	Application Association
AARE	A-Associate Response – an APDU of the ACSE

Abbreviation	Meaning
AARQ	A-Associate Request – an APDU of the ACSE
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
BER	Basic Encoding Rules
BD	Block Data
BN	Block Number
BNA	Block Number Acknowledged
BS	Bit string
BTS	Block Transfer Streaming
BTW	Block Transfer Window
CA	Certification Authority
CF	Control Function
CL	Connectionless
class_id	COSEM interface class identification code
CMP	Certificate Management Protocol. Refer to RFC 4210.
CO	Connection-oriented
COSEM	Companion Specification for Energy Metering
COSEM_on_IP	The TCP-UDP/IP based COSEM communication profile
CRC	Cyclic Redundancy Check
CRL	Certificate revocation list. Refer to RFC 5280.
CSR	Certificate Signing Request
DCE	Data Communication Equipment (communications interface or modem)
DCS	Data Collection System
DISC	Disconnect (a HDLC frame type)
DLMS	Device Language Message Specification
DM	Disconnected Mode (a HDLC frame type)
DSA	Digital Signature Algorithm specified in FIPS PUB 186-4
DSAP	Data Link Service Access Point
DSO	Energy Distribution System Operator
DTE	Data Terminal Equipment (computers, terminals or printers)
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman key agreement protocol

Abbreviation	Meaning
ECDSA	Elliptic Curve Digital Signature Algorithm specified in ANSI X9.62 and FIPS PUB 186-4
ECP	Elliptic Curve Point
EUI-64	64-bit Extended Unique Identifier
FCS	Frame Check Sequence
FDDI	Fibre Distributed Data Interface
FE	Field Element (in relation with public key algorithms)
FIPS	Federal Information Processing Standard
FRMR	Frame Reject (a HDLC frame type)
FTP	File Transfer Protocol
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
GSM	Global System for Mobile communications
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
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)
MIB	Management Information Base
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
N(R)	Receive sequence Number
N(S)	Send sequence Number
NDM	Normal Disconnected Mode
NIST	National Institute of Standards and Technology
NNAP	Neighbourhood Network Access Point
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
PAR	Positive Acknowledgement with Retransmission
PDU	Protocol data unit
PhL	Physical Layer
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
PSTN	Public Switched Telephone Network
RA	Registration Authority
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)
RR	Receive Ready (a HDLC frame type)

Abbreviation	Meaning
RSA	Algorithm developed by Rivest, Shamir and Adelman; specified in ANS X9.31 and PKCS #1.
SAP	Service Access Point
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)
STR	Streaming
tbsCertificate	To be signed certificate
TCP	Transmission Control Protocol
TDEA	Triple Data Encryption Algorithm
TL	Transport Layer
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)
UNC	Unbalanced operation Normal response mode Class
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
See also list of abbreviations specific to a cryptographic algorithm in the relevant clauses.	

3.5 Symbols related to the Galois/Counter Mode

Symbol	Meaning
A	Additional Authenticated Data, AAD
AK	Authenticaiton 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
len(X)	The bit length of the bit string X.
LEN(X)	The octet length of the octet string X.
P	Plaintext
SC	Security Control Byte
SH	Security Header
Sys-T	System title
T	Authentication tag
t	The bit length of the authentication tag. NOTE This is the same as len(T)
X II Y	Concatenation of two strings, X and Y.

3.6 Symbols related 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.

3.7 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

3.8 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 _{ELL}	CI field introducing the extended link layer (wireless M-Bus)	M-Bus
CI Field	Control Information field	M-Bus
CI _{TL}	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 metering 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 "metering devices" is an abstraction; consequently "metering device" may be any type of device for which this abstraction is suitable.

The COSEM object model is specified in DLMS UA 1000-1, the "Blue Book". The COSEM objects provide a view of the functionality of metering 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:

- metering devices can be accessed by various parties: clients and third parties;
- mechanisms to control access to the resources of the metering 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 metering site, there may be single or multiple metering devices. In the case of multiple metering devices at a metering site, a single access point can be made available;
- data exchange may take place either remotely or locally. Depending on the capabilities of the metering 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 meters 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. Their use is outlined in this clause and further developed in other clauses.

Application functions of metering 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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	53/500
-----------------------	------------	------------------------	--------

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 metering devices is based on the client/server model where data collection systems play the role of the client and metering 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.

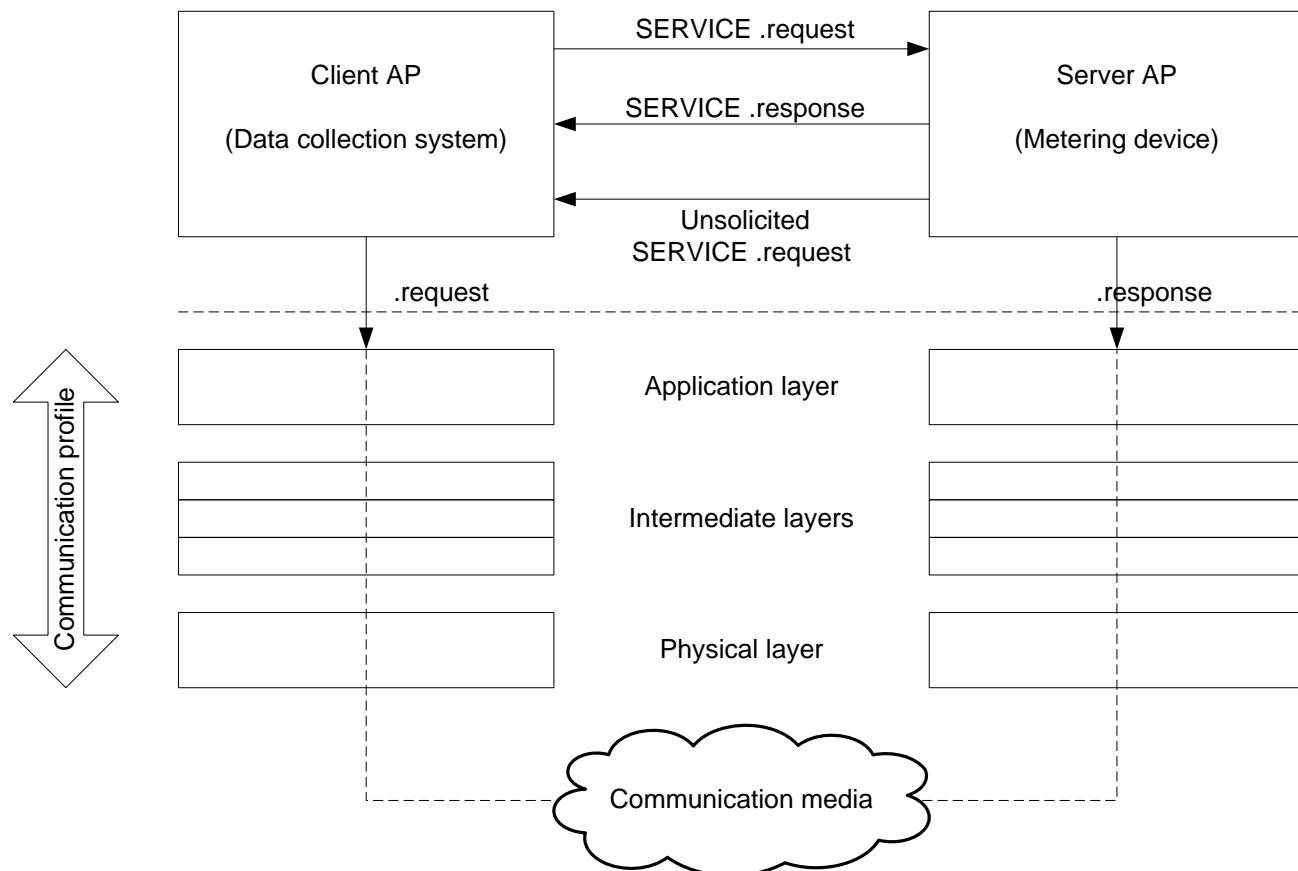


Figure 2 – Client–server model and communication protocols

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.

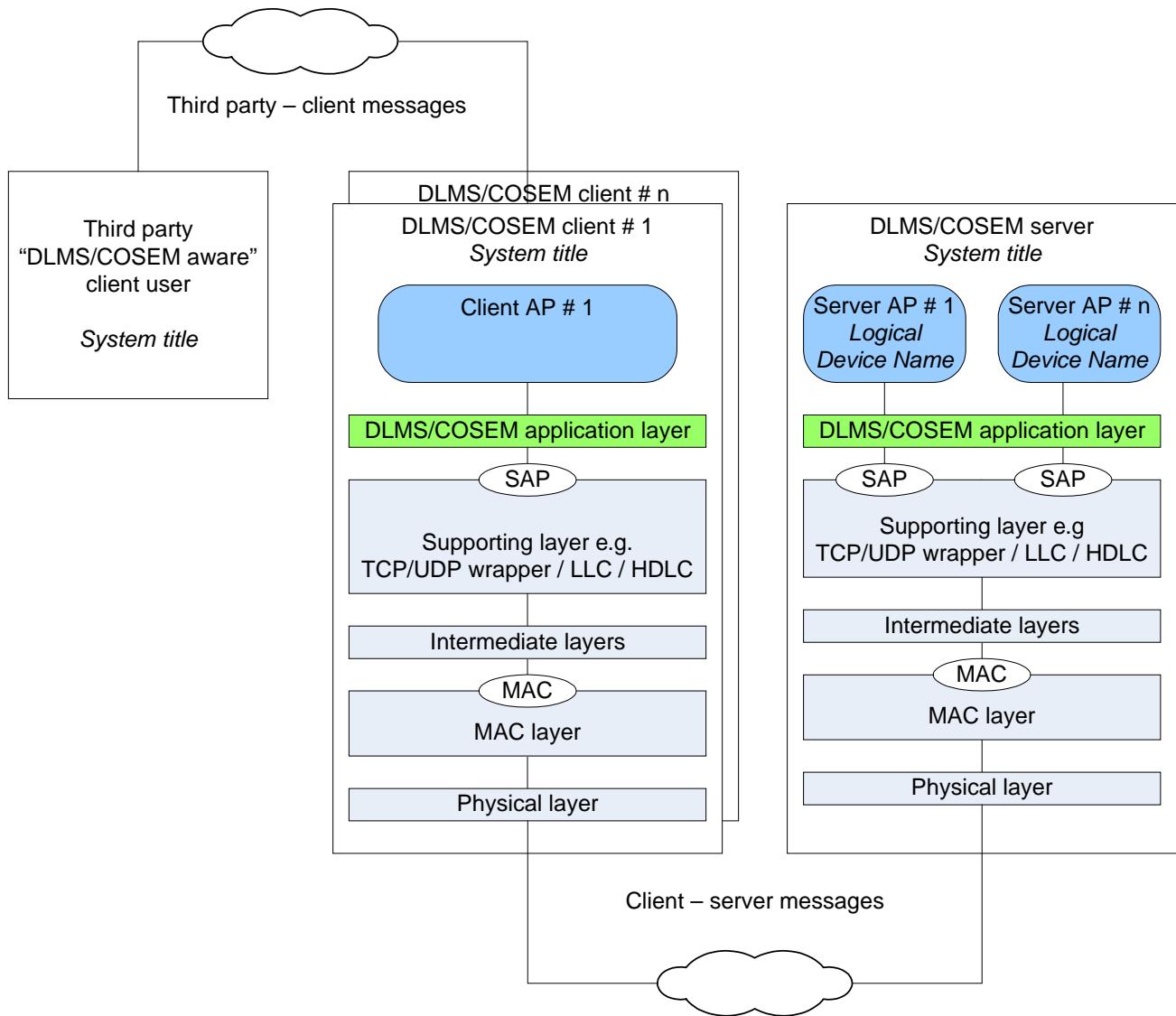


Figure 3 – Naming and addressing in DLMS/COSEM

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 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/COSEM 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, an 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 Ed. 12.1:2015 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

Client SAPs	
No-station	0x00
Client Management Process / CIASE ¹	0x01
Public Client	0x10
<i>Open for client AP assignment</i>	0x02 ... 0x0F 0x11 and up
Server SAPs	
No-station / CIASE ¹	0x00
Management Logical Device	0x01
Reserved for future use	0x02...0x0F
<i>Open for server SAP assignment</i>	0x10 and up
All-station (Broadcast)	Communication profile specific

¹ In the case of the DLMS/COSEM S-FSK PLC profile, see 10.4.

NOTE Depending on the supporting 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 that may be 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 leading (i.e., the 3 leftmost) octets should hold the three-letter manufacturer ID¹. This is the same as the leading three octets of the Logical Device Name, see 4.3.5. The remaining 5 octets shall ensure uniqueness.

NOTE It can be derived for example from the last 12 digits of the manufacturing number, up to 999 999 999 999. 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 as such 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.

Before the cryptographic security algorithms can be used – this requires a ciphered application context – the peers have to exchange system titles. The following possibilities are available:

- 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 10.4.5;
- in all communication profiles, system titles may be exchanged during AA establishment using the COSEM-OPEN service, see 9.3.2, carried the AARQ / AARE APDU. 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;

¹ Administered by the FLAG Association in co-operation with the DLMS UA.

- 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. 12.1:2015 4.4.7.

In the case of broadcast communication, only the client sends the system title to the server.

4.3.5 Logical Device Name

Logical Device Name (LDN) shall be as specified in DLMS UA 1000-1 Ed. 12.1:2015 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 meter. It is specified in DLMS UA 1000-1 Ed. 12.1:2015 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.

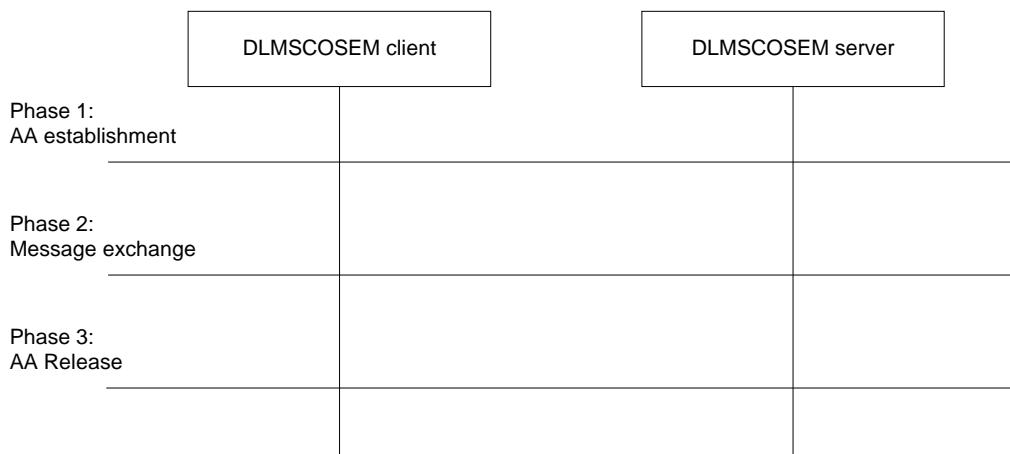


Figure 4 – A complete communication session in the CO environment

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.

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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	58/500
-----------------------	------------	------------------------	--------

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.

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 meter reader, meter service / communication service / energy provider, manufacturer, end user etc.

4.6 Messaging patterns

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

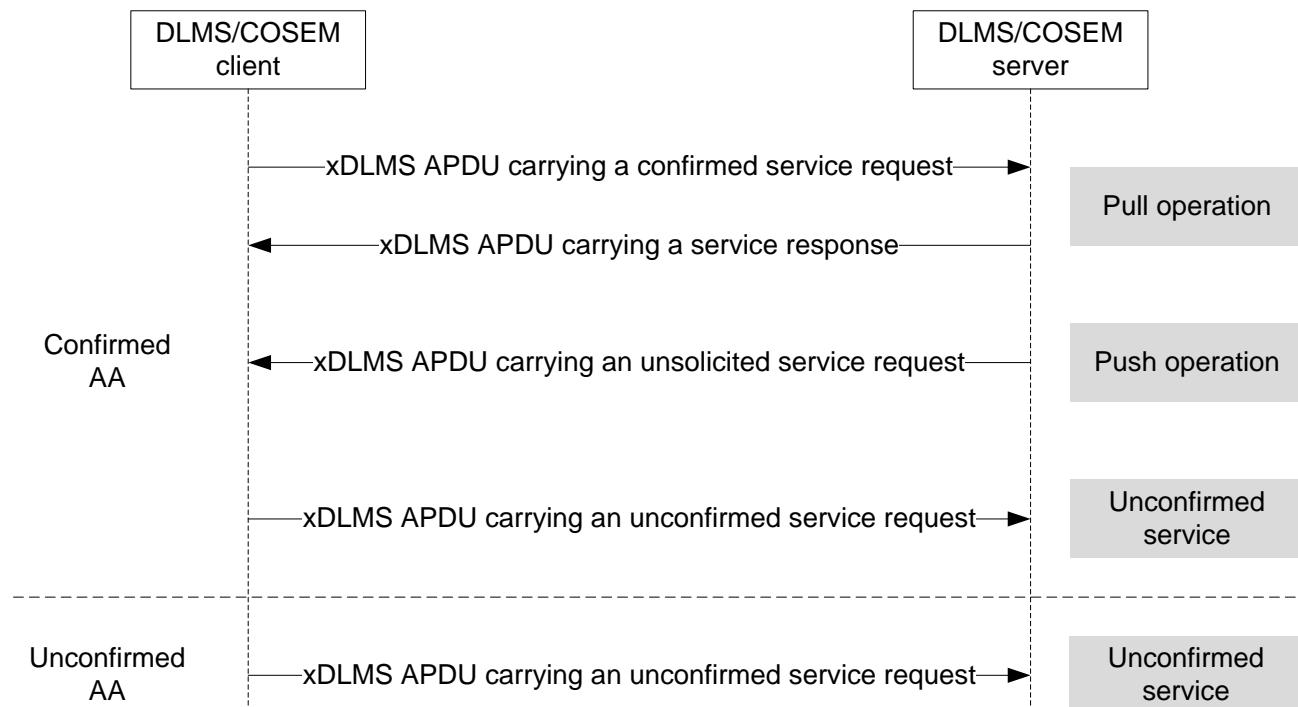


Figure 5 – DLMS/COSEM messaging patterns

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.

4.7 Data exchange between third parties and DLMS/COSEM servers

Third parties – that are outside the DLMS/COSEM 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 66.

NOTE 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.

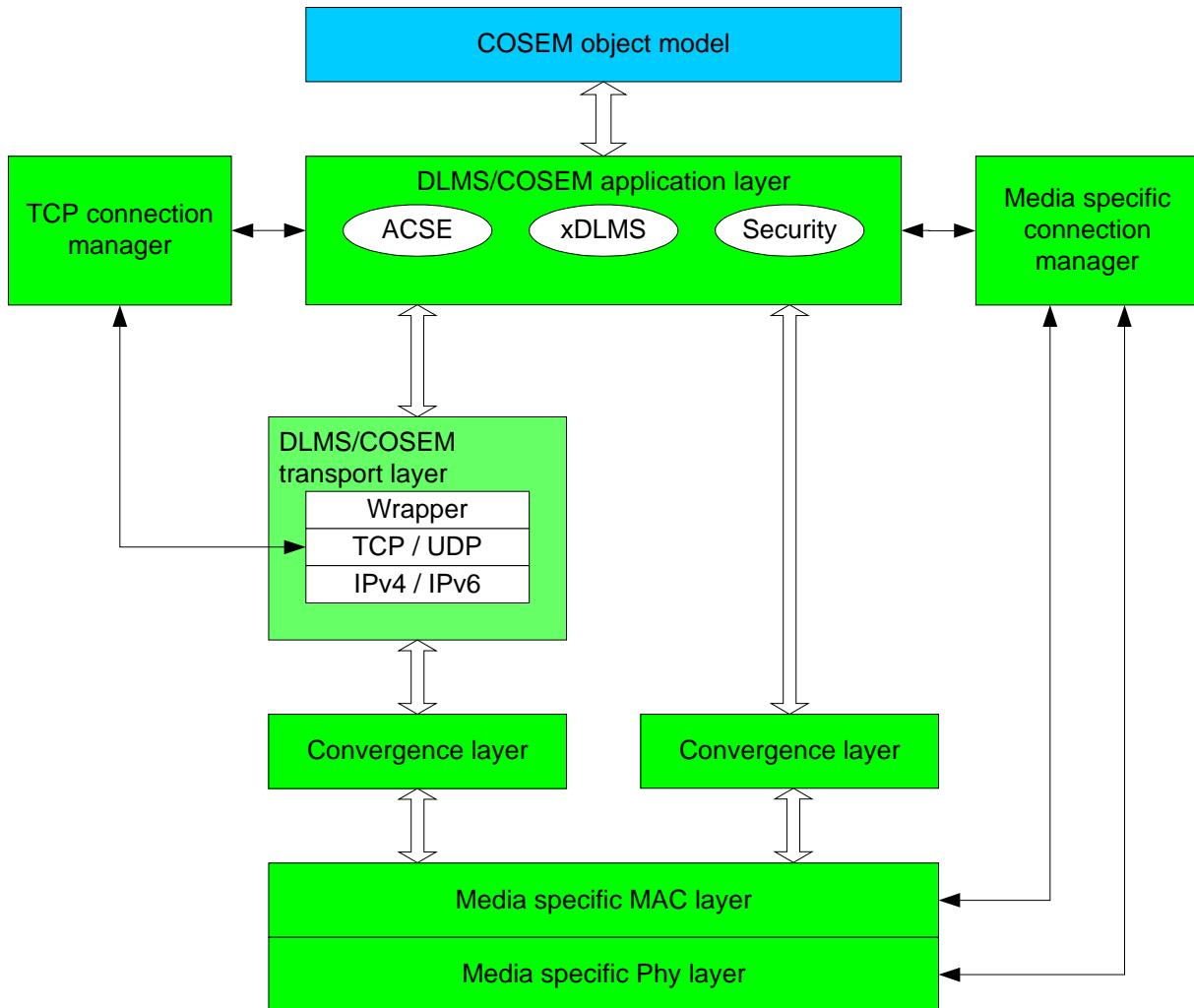
Communication profiles comprise a number of protocol layers. Each layer has a distinct task and provides services to its upper layer and uses services of its supporting layer(s). The client and server COSEM APs use the services of the highest protocol layer, that of the DLMS/COSEM AL. This is the only 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 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 transport layer;
- the media specific physical and MAC layers; and
- the connection managers.

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, is specified in 10.2;
- The TCP-UDP/IP based communication profiles (COSEM_on_IP), is specified in 10.3;
- The S-FSK PLC profile, is specified in 10.4;
- The wired and wireless M-Bus profile is specified in 10.5.

4.9 Model of a DLMS/COSEM metering system

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

Metering 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 metering equipment as these are seen through its communication interfaces. The various functions are modelled using COSEM objects.

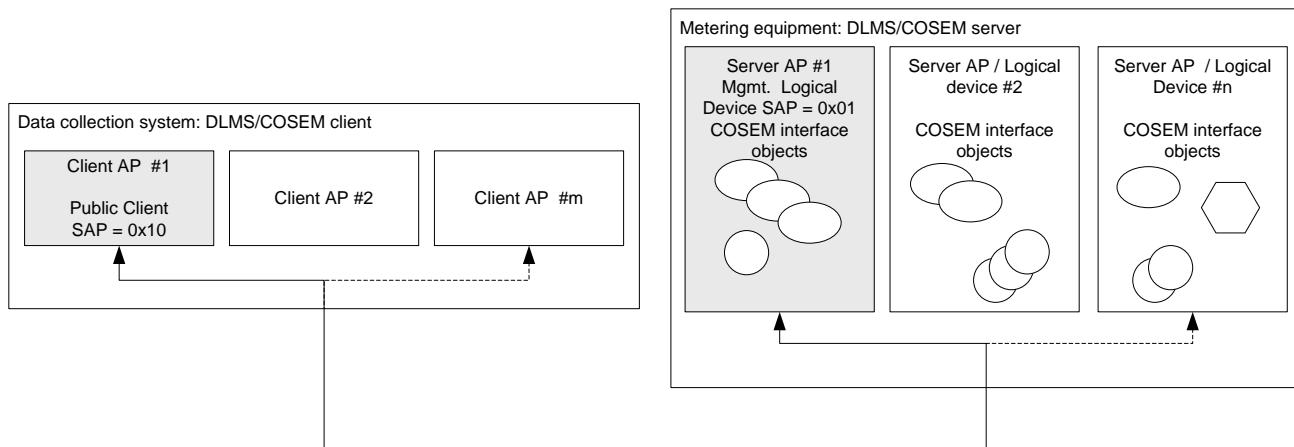


Figure 7 – Model of a DLMS/COSEM metering system

Data collection systems are modelled as a set of client APs. Each client AP may have different roles and access rights, granted by the metering equipment.

NOTE The application processes may be hosted by one or several physical devices.

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 Ed. 12.1:2015, DLMS UA 1000-1 4.1.7 and 4.1.8.

4.10 Model of DLMS/COSEM servers

Figure 8 shows the model of two DLMS/COSEM 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.

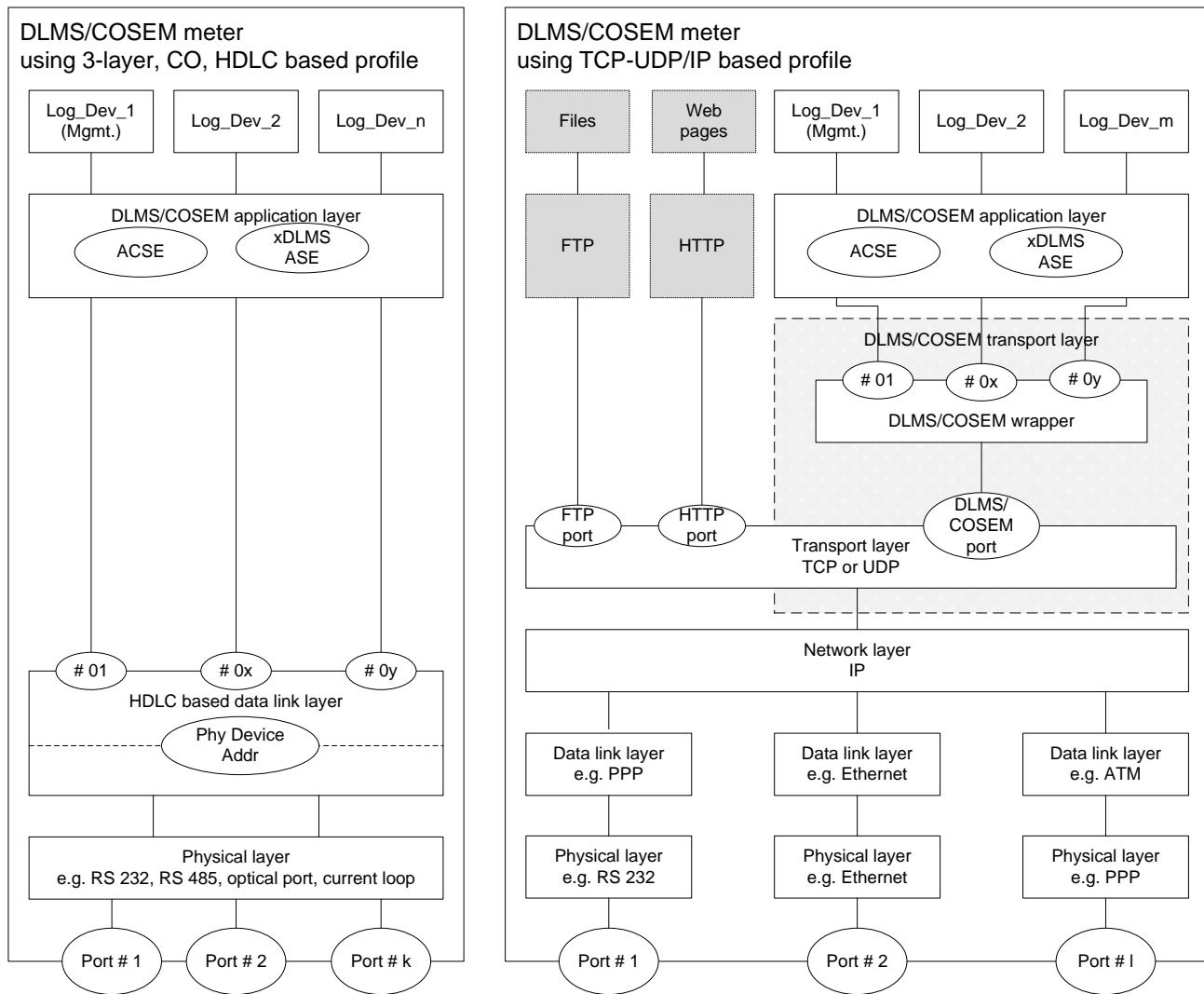
The metering equipment on the left hand side comprises "n" logical devices and supports the 3-layer, CO, HDLC based communication profile.

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 metering devices at a given metering 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 metering 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/COSEM server model**

Through the wrapper, the DLMS/COSEM AL is bound to a TCP or UDP port number, 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 layer is 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.).

Obviously, in a single server it is possible to implement several protocol stacks, 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/COSEM client show below.

4.11 Model of a DLMS/COSEM client

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

The model of the client – obviously – 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 DLMS/COSEM TL, 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 layer, which in turn use the services of its supporting 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).

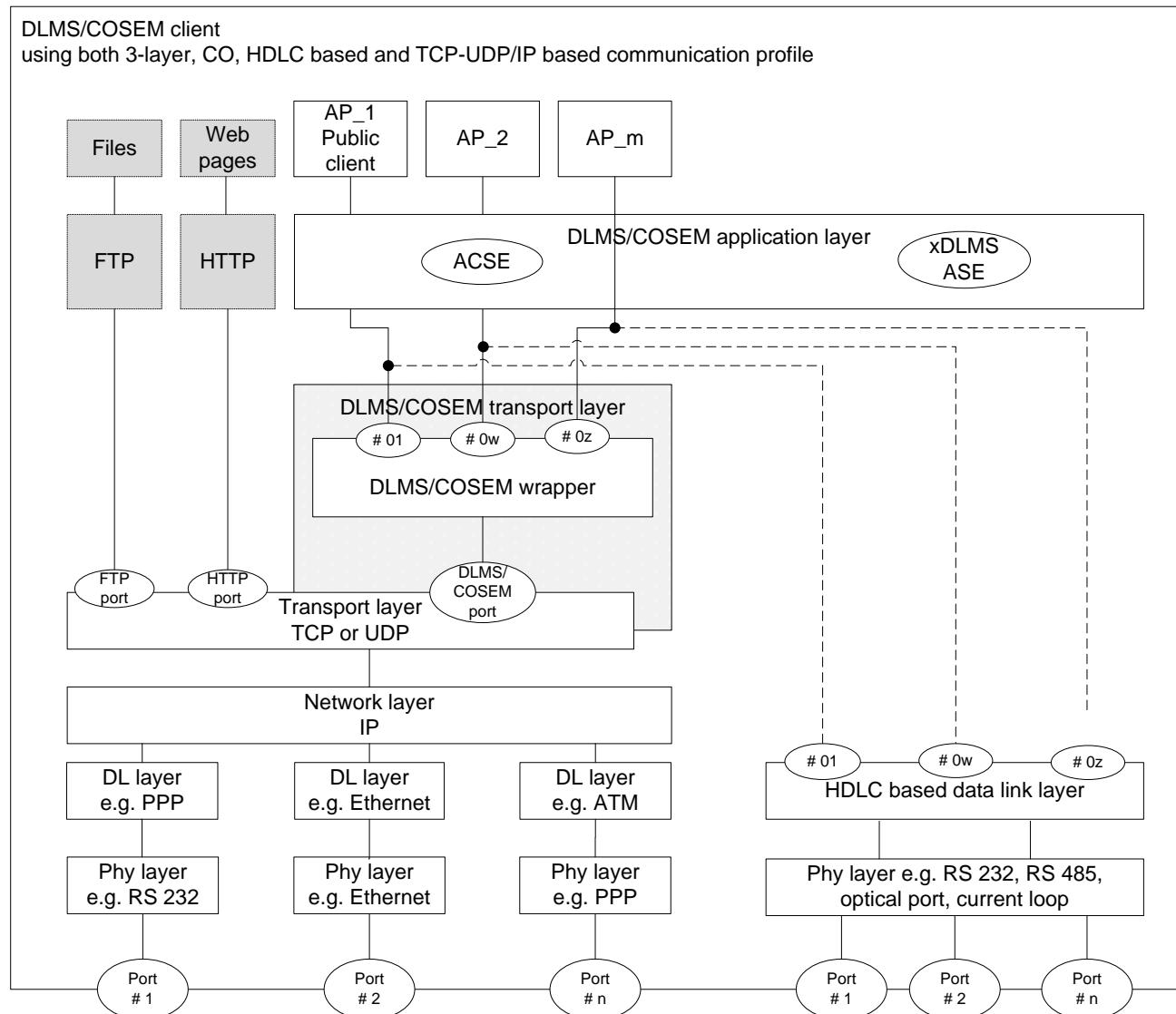


Figure 9 – Model of a DLMS/COSEM client using multiple protocol stacks

As explained, 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 to support 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 metering of all kinds of energy, 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 must 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 meter 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 – metering equipment. 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 meter 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 meter, and then send the necessary configuration information (for example tariff schedules and installation specific parameters) to the meter. With this, the meter 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 metering 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 10.4.4.2. 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.

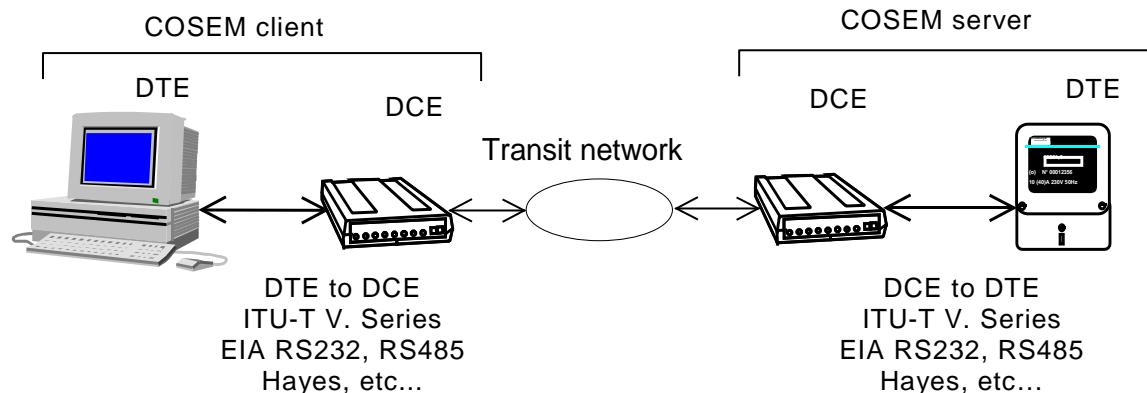


Figure 10 – Typical PSTN configuration

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 metering 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 metering 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.

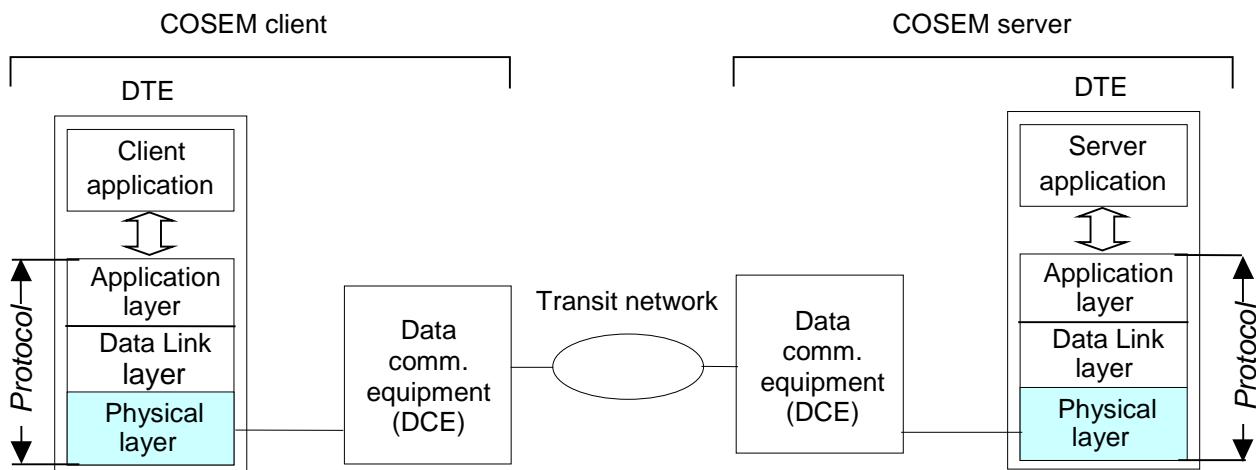


Figure 11 – The location of the physical layer

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.

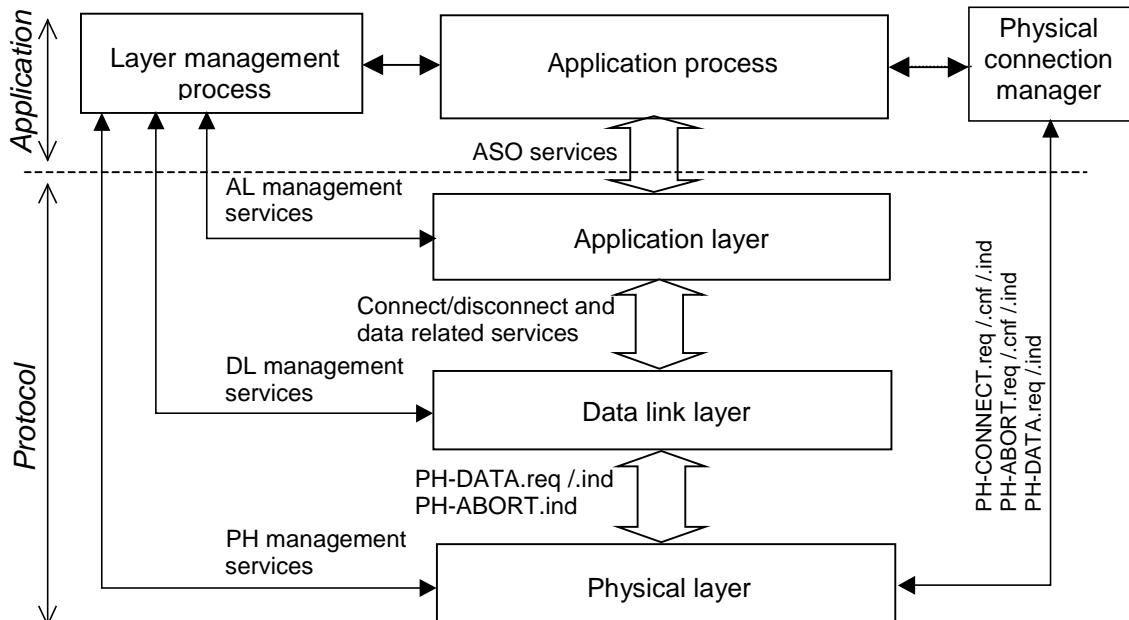
Layer management services are used by or provided for the layer management process, which is part of the AP. Some examples are given below:

- 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

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

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

The PhConnType parameter 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.

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

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

PH-CONNECT.indication ()

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*Function*

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

PH-CONNECT.confirm (Result,
PhConnCnfParams
)

The Result parameter 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.

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***Function*

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

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.

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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	69/500
-----------------------	------------	------------------------	--------

5.2.3.2.2 PH-DATA.indication

Function

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

Semantics of the service primitive

The semantics of this primitive is as follows:

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

The Data parameter carries the received byte as received by the local PhL entity.

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

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

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

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

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

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

The Result parameter carries the result of the physical disconnection attempt.

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

Function

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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	70/500
-----------------------	------------	------------------------	--------

Semantics of the service primitive

The primitive shall provide parameters as follows:

PH-ABORT.indication ()

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.

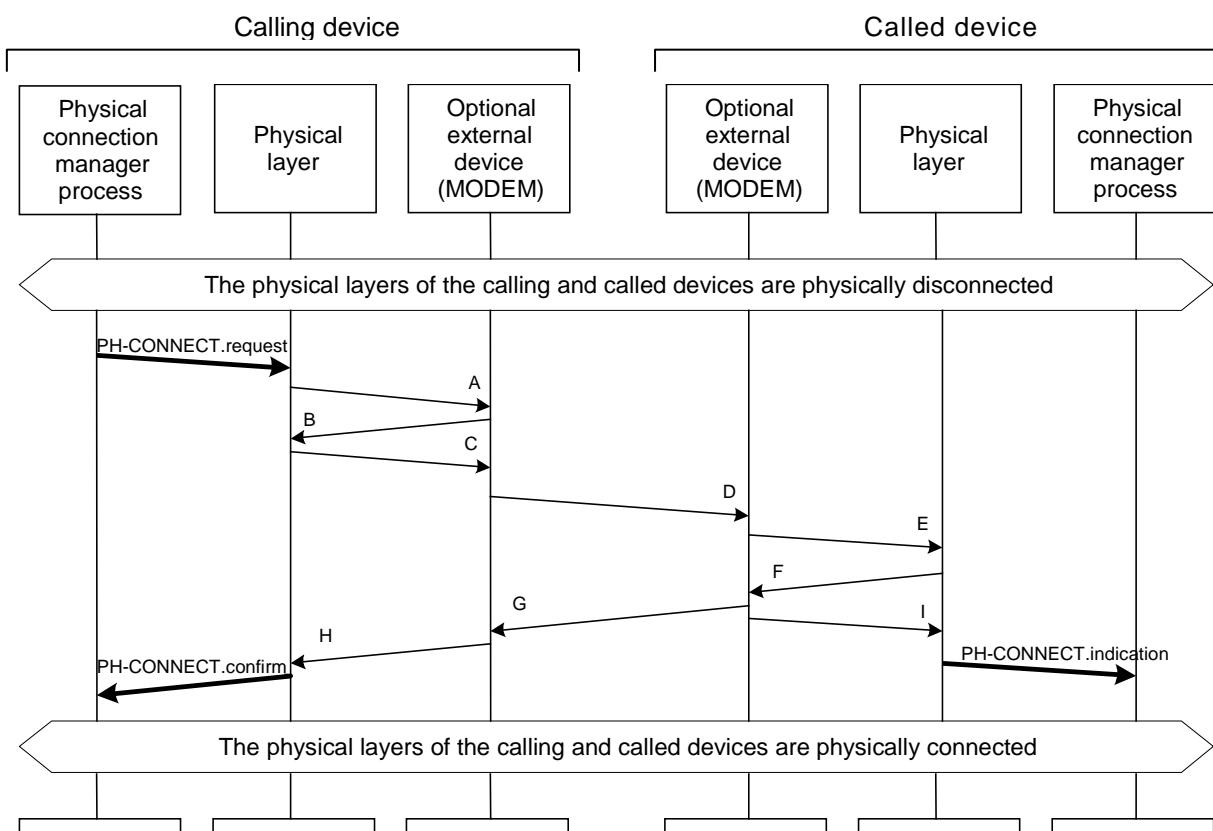


Figure 13 – MSC for physical connection establishment

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.

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 -- 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.

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

³ As specified in ANSI C12.21.

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.

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.

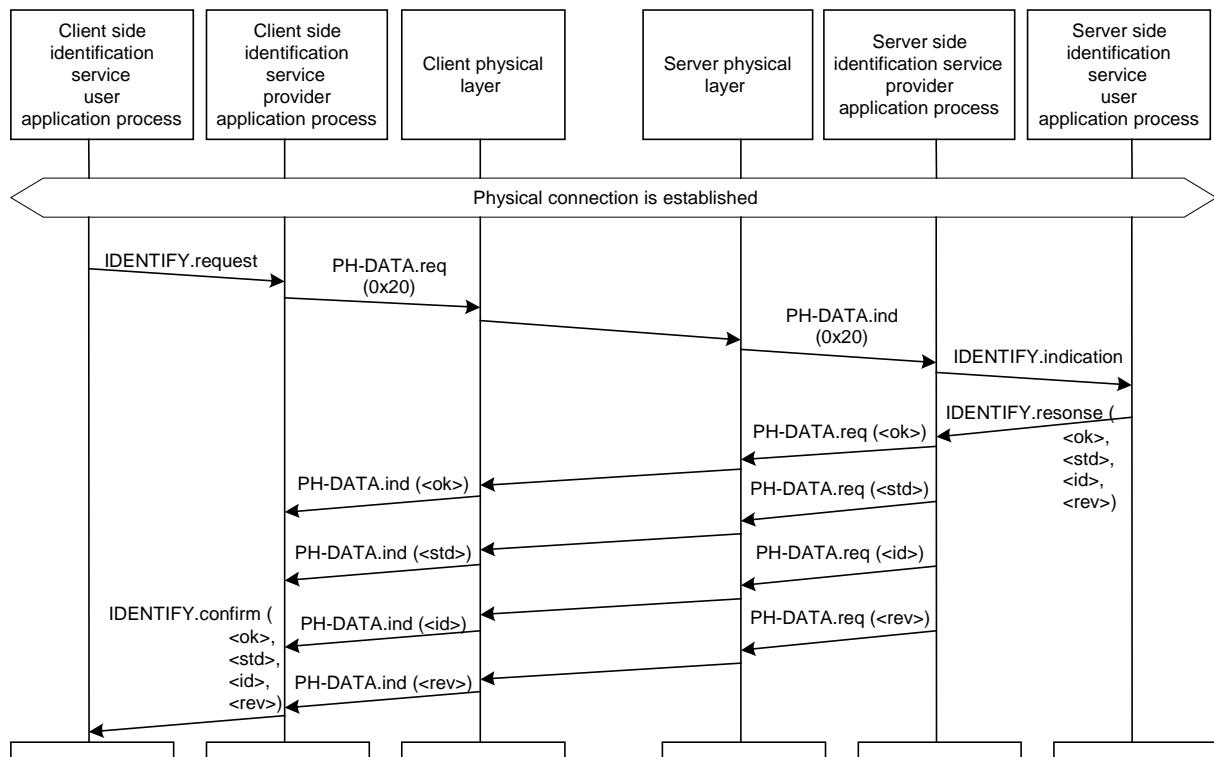


Figure 14 – MSC for IDENTIFY.request / .response message exchange

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

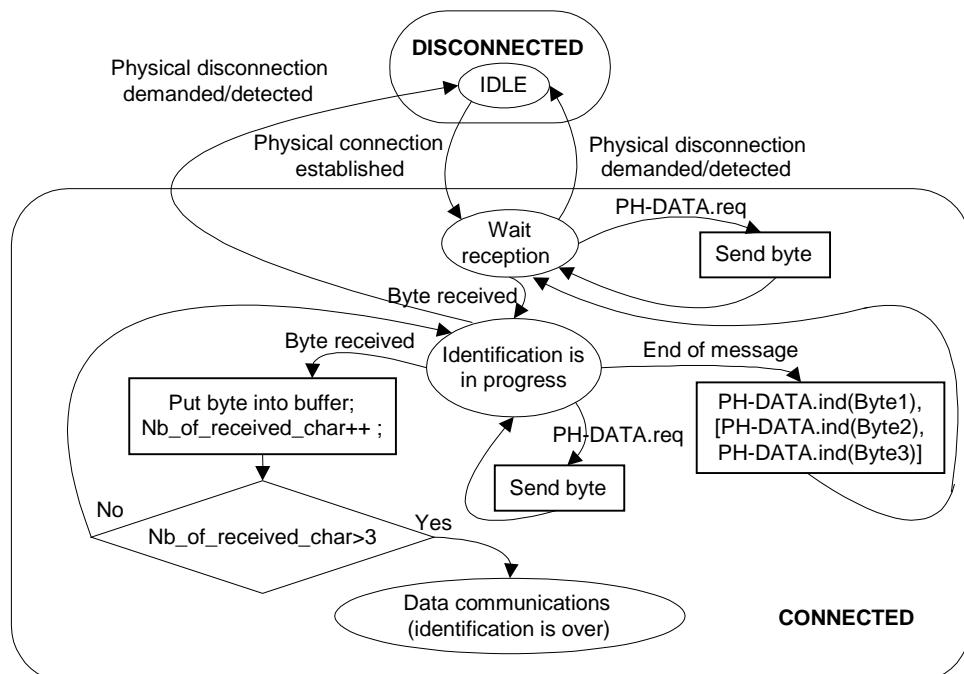


Figure 15 – Handling the Identification service at the server side

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.

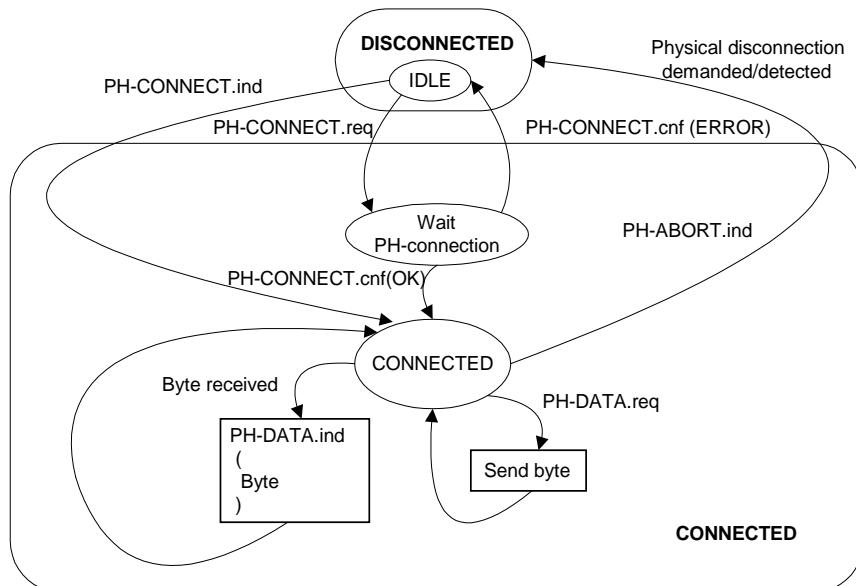


Figure 16 – Partial state machine for the client side physical layer

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 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.

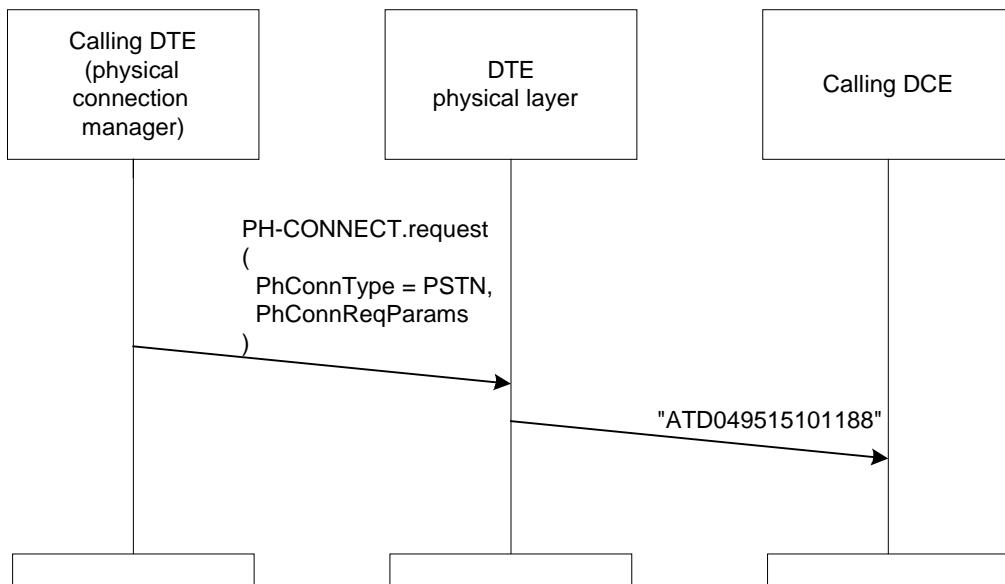
5.4.2 Physical layer services and related message exchanges

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.

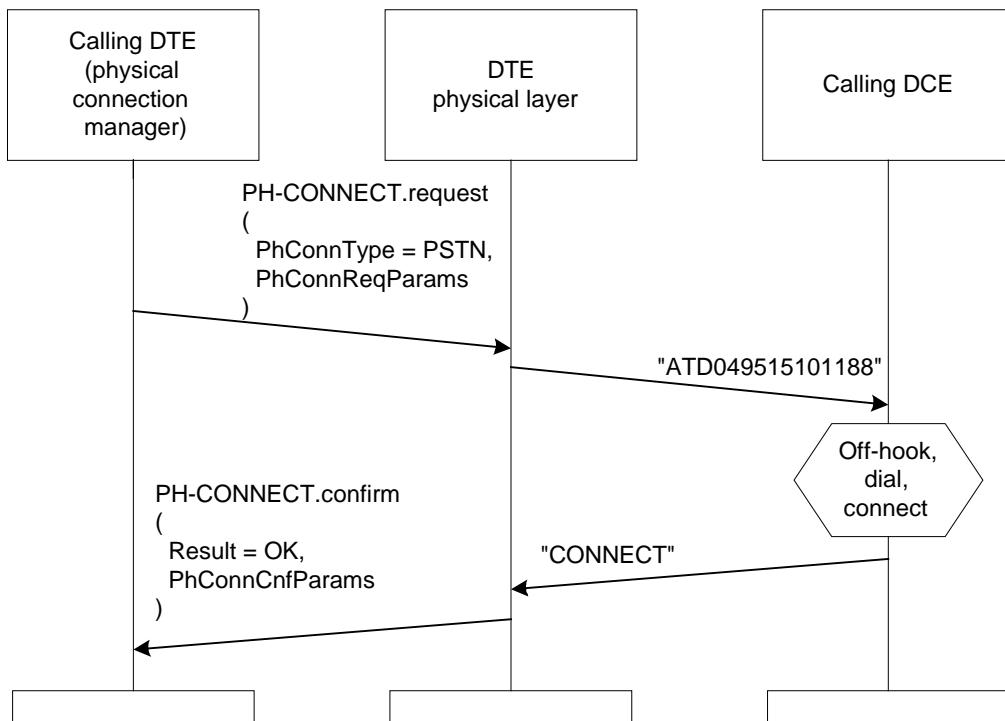


Figure 18 – Physical connection establishment at the CALLING station

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.

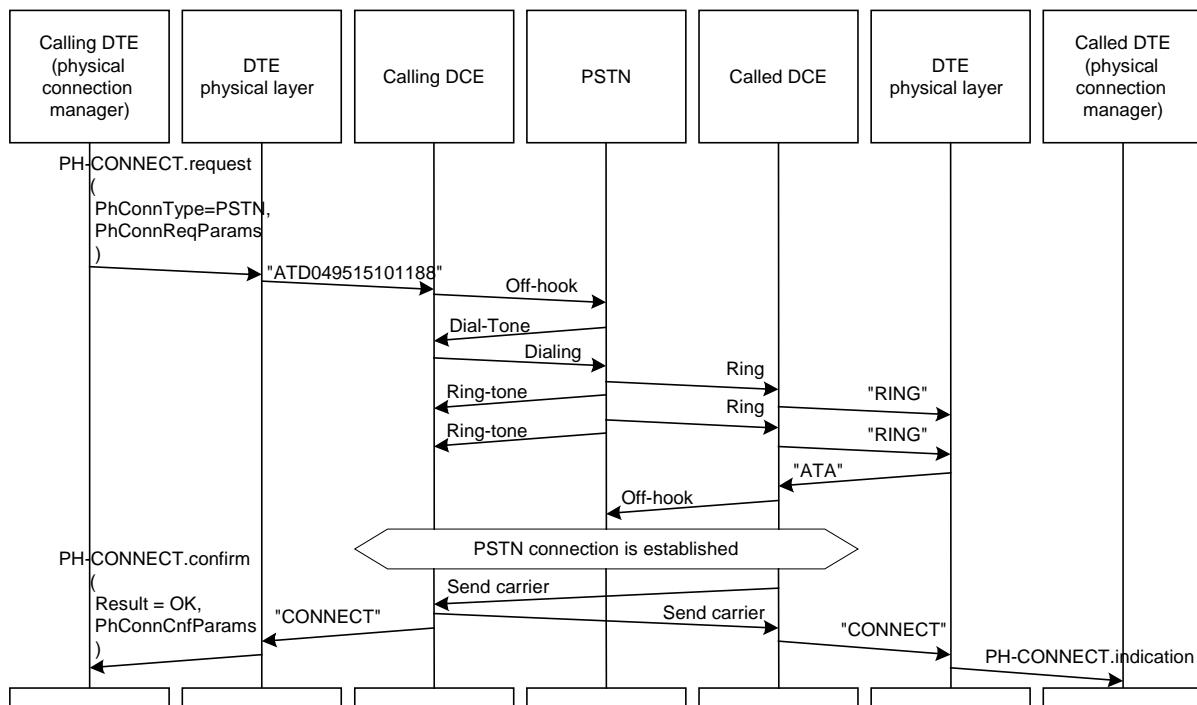


Figure 19 – MSC for physical connection establishment

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.

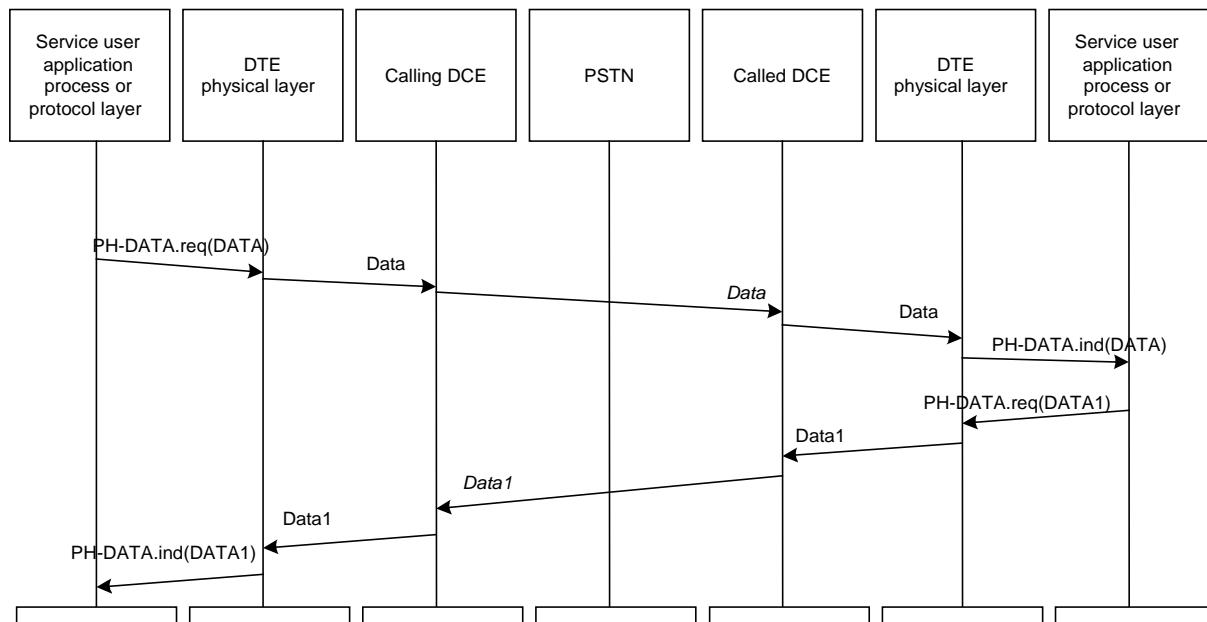


Figure 20 – Data exchange between the calling and called stations

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.

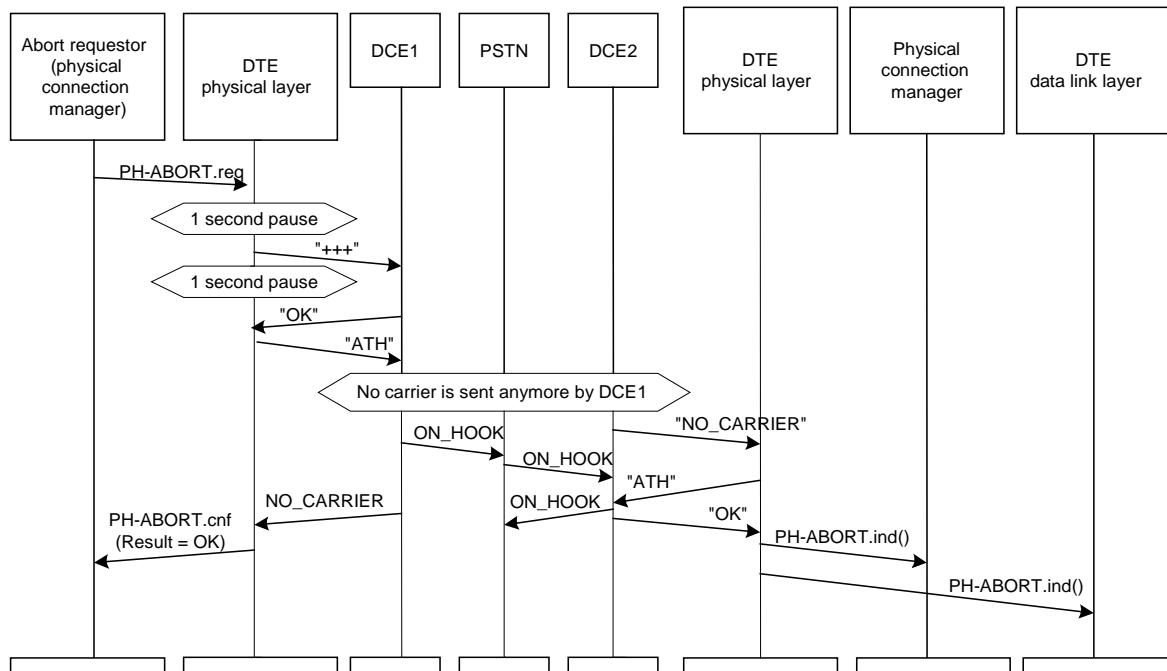


Figure 21 – MSC for a physical disconnection

6 Direct Local Connection (excerpt)

6.1 Introduction

This chapter 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, Clause 6.3.14⁴.

6.3 Overview

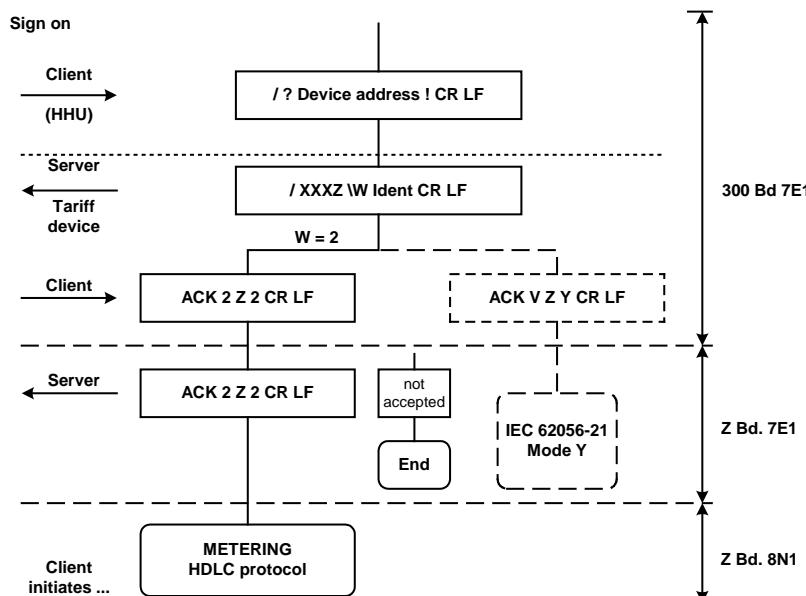


Figure 22 – Entering protocol mode E (HDLC)

⁴ 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.

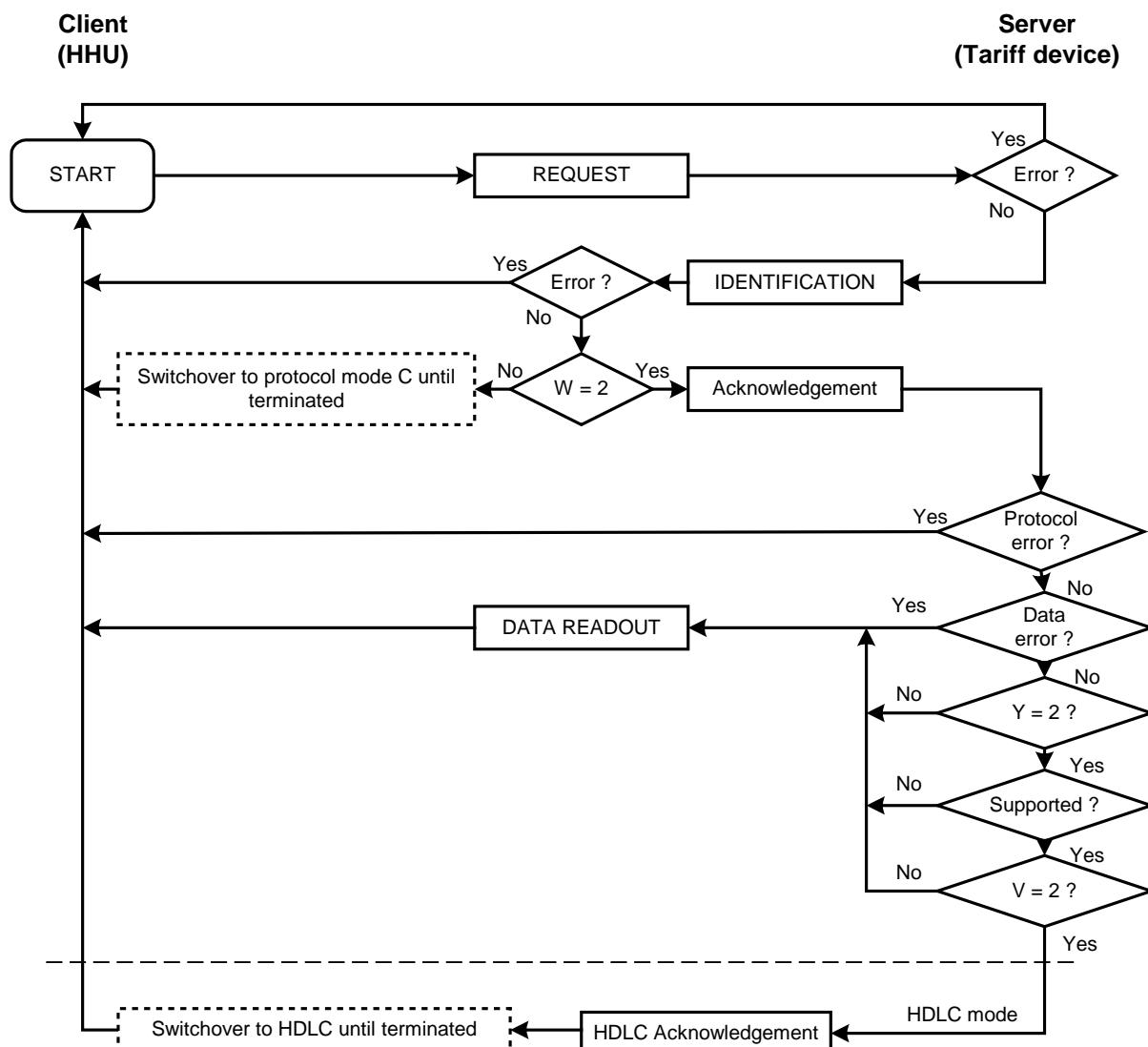


Figure 23 – Flow chart and switchover to METERING HDLC in protocol mode E

6.5 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.

6.6 Physical layer – Introduction

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

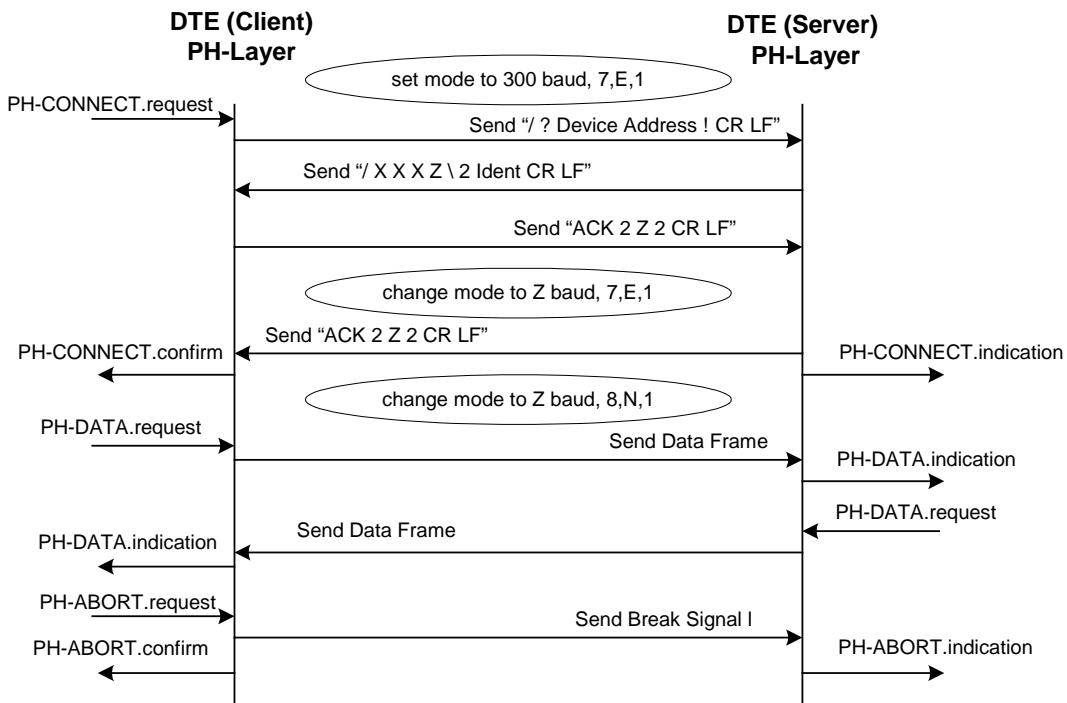


Figure 24 – Physical layer primitives

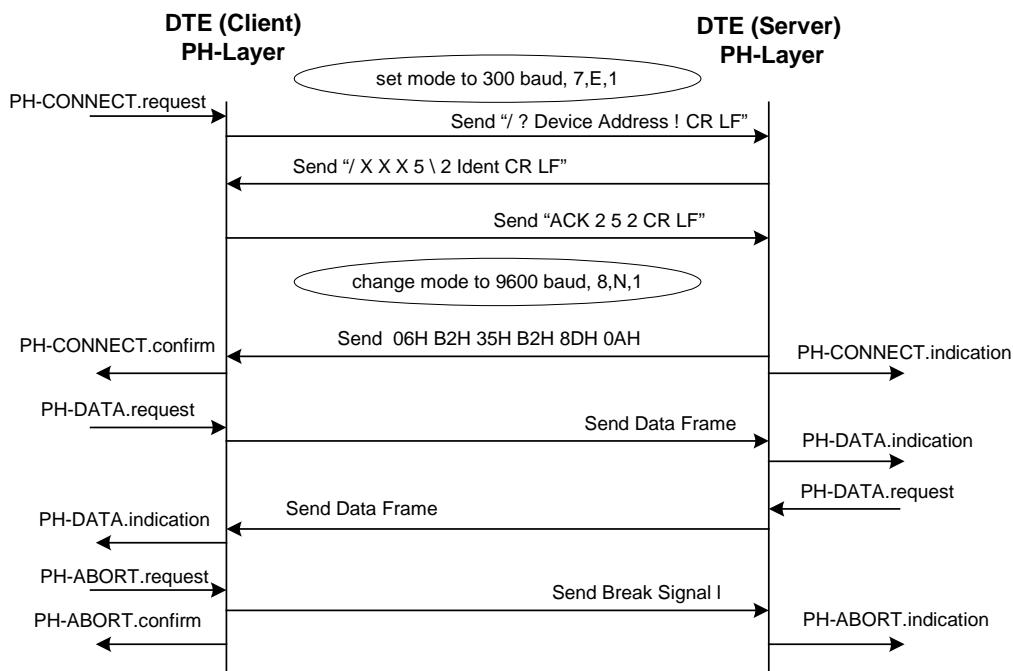


Figure 25 – Physical layer primitives, simplified example with one mode change only

6.7 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.

6.8 Data link layer

See Clause 8.

7 DLMS/COSEM transport layer for IP networks

7.1 Scope

This Clause 7 specifies a connection-less and a connection oriented transport layer (TL) for DLMS/COSEM 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).

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

Clause 7.5 shows how the OSI-style TL services can be converted to and from UDP and TCP function calls.

7.2 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.

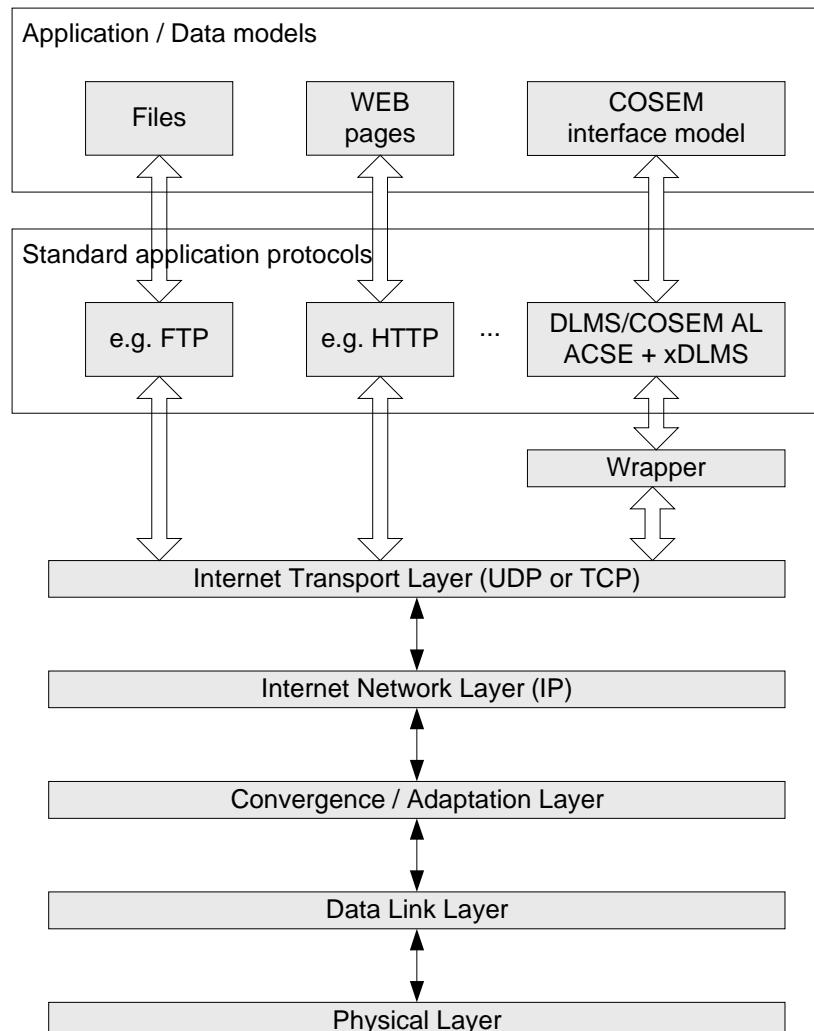


Figure 26 – DLMS/COSEM as a standard Internet application protocol

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.

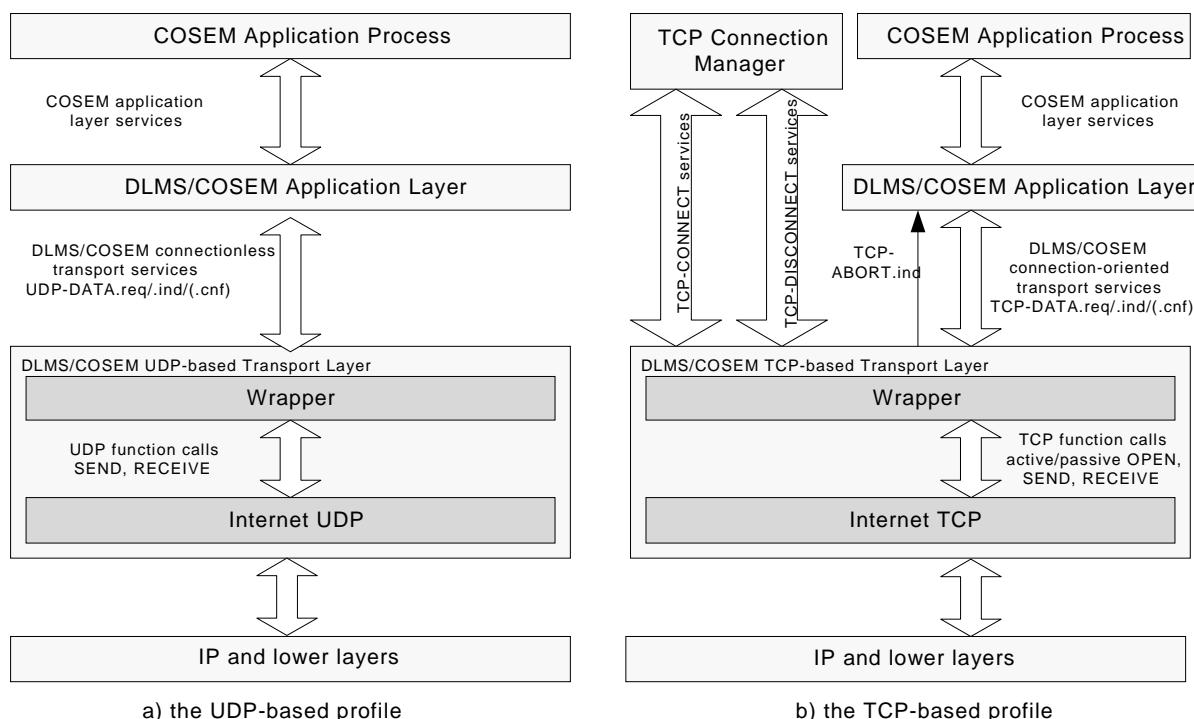


Figure 27 – Transport layers of the DLMS/COSEM_on_IP profile

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.

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

7.3.1 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.

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

7.3.2.1 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.

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.

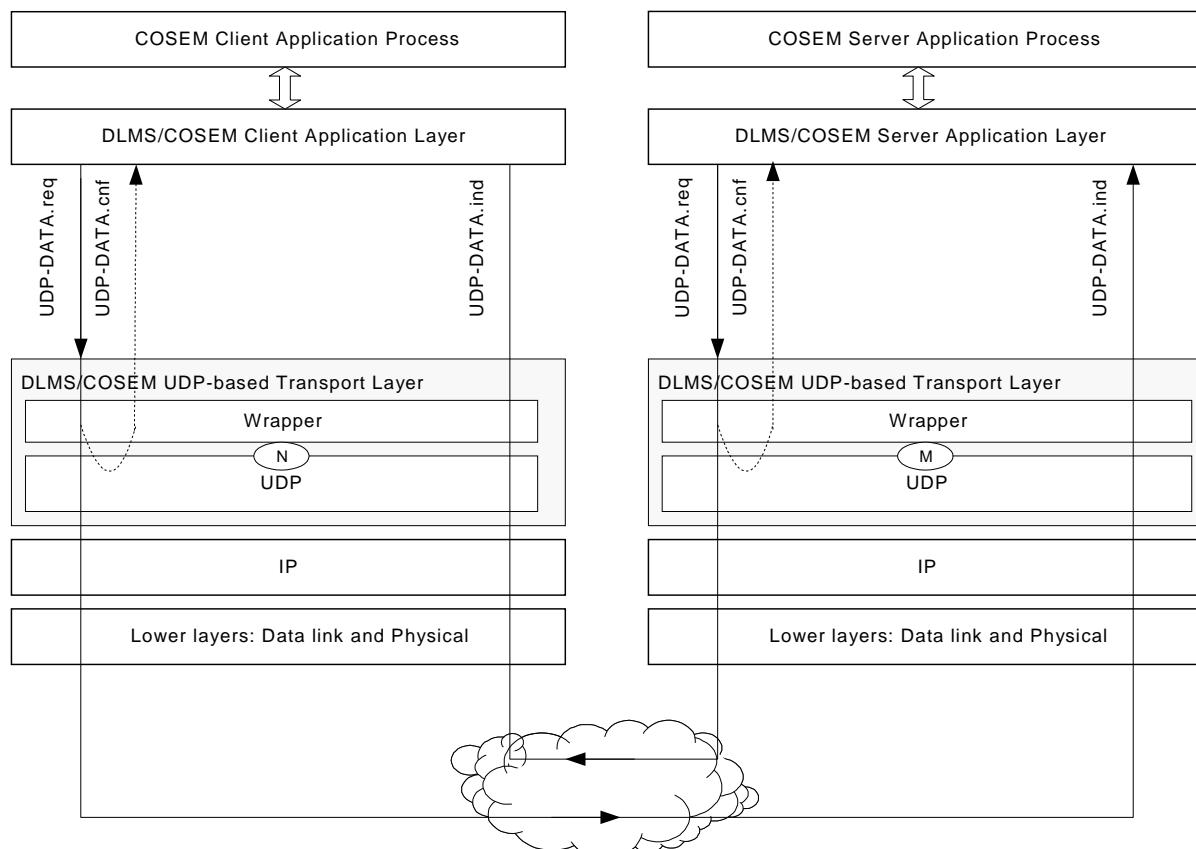


Figure 28 – Services of the DLMS/COSEM connection-less, UDP-based transport layer**7.3.2.2 The UDP-DATA service****7.3.2.2.1 UDP-DATA.request***Function*

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

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
)
```

The Local_wPort, Local_UDP_Port and Local_IP_Address parameters 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.

The Data_Length parameter indicates the length of the Data parameter in bytes.

The Data parameter contains the xDLMS APDU to be transferred to the peer AL.

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 7.3.3.2, as DATA. The UDP sublayer shall transmit the WPDU to the peer wrapper sublayer as described in STD0006.

7.3.2.2.2 UDP-DATA.indication*Function*

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

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,
```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	88/500
-----------------------	------------	------------------------	--------

```

    Data
)

```

The Local_wPort, Local_UDP_Port and Local_IP_Address parameters 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.

The Data_Length parameter indicates the length of the Data parameter in bytes.

The Data parameter contains the xDLMS APDU received from the peer AL.

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.

7.3.2.2.3 UDP-DATA.confirm

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

```

UDP-DATA.confirm (
    Local_wPort,
    Remote_wPort,
    Local_UDP_Port,
    Remote_UDP_Port,
    Local_IP_Address,
    Remote_IP_Address,
    Result
)

```

The Local_wPort, Remote_wPort, Local_UDP_Port, Remote_UDP_Port, Local_IP_Address and Remote_IP_Address parameters 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).

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.

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

7.3.3.1 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	89/500
-----------------------	------------	------------------------	--------

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.

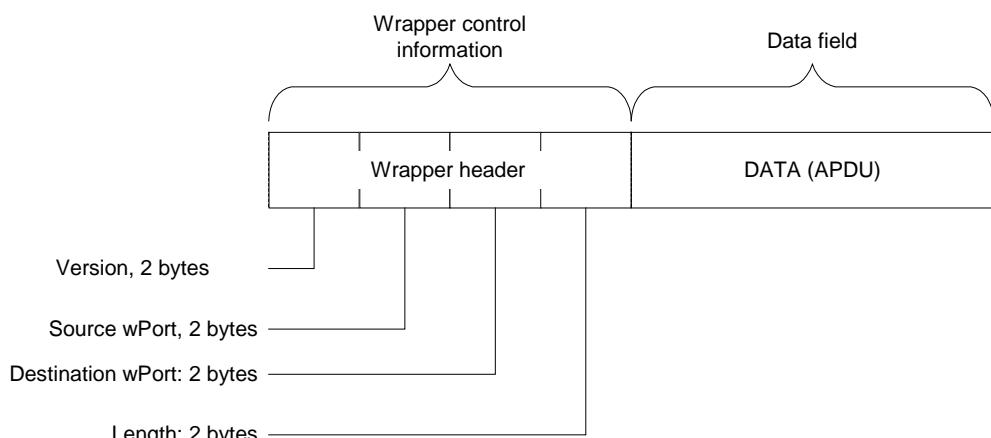
7.3.3.2 The wrapper protocol data unit (WPDU)

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.

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).

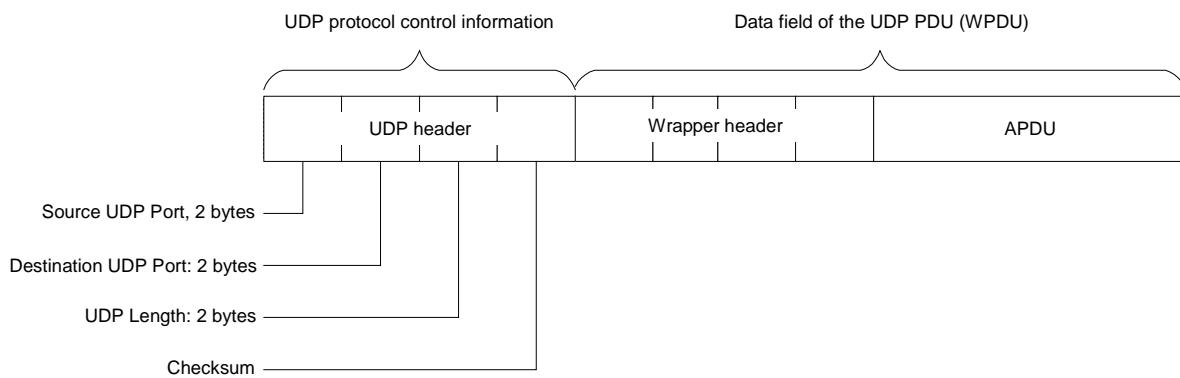


NOTE The maximum length of the APDU should be eight bytes less than the maximum length of the UDP datagram.

Figure 29 – The wrapper protocol data unit (WPDU)

7.3.3.3 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.

7.3.3.4 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

Client side reserved addresses	
	Wrapper Port Number
No-station	0x0000
Client Management Process	0x0001
Public Client	0x0010
<i>Open for client SAP assignment</i>	0x02 ... 0x0F
	0x11... 0xFF
Server side reserved addresses	
	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

7.3.3.5 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.

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

7.4.1 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 retransmission 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 7.5.

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

7.4.2.1 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.

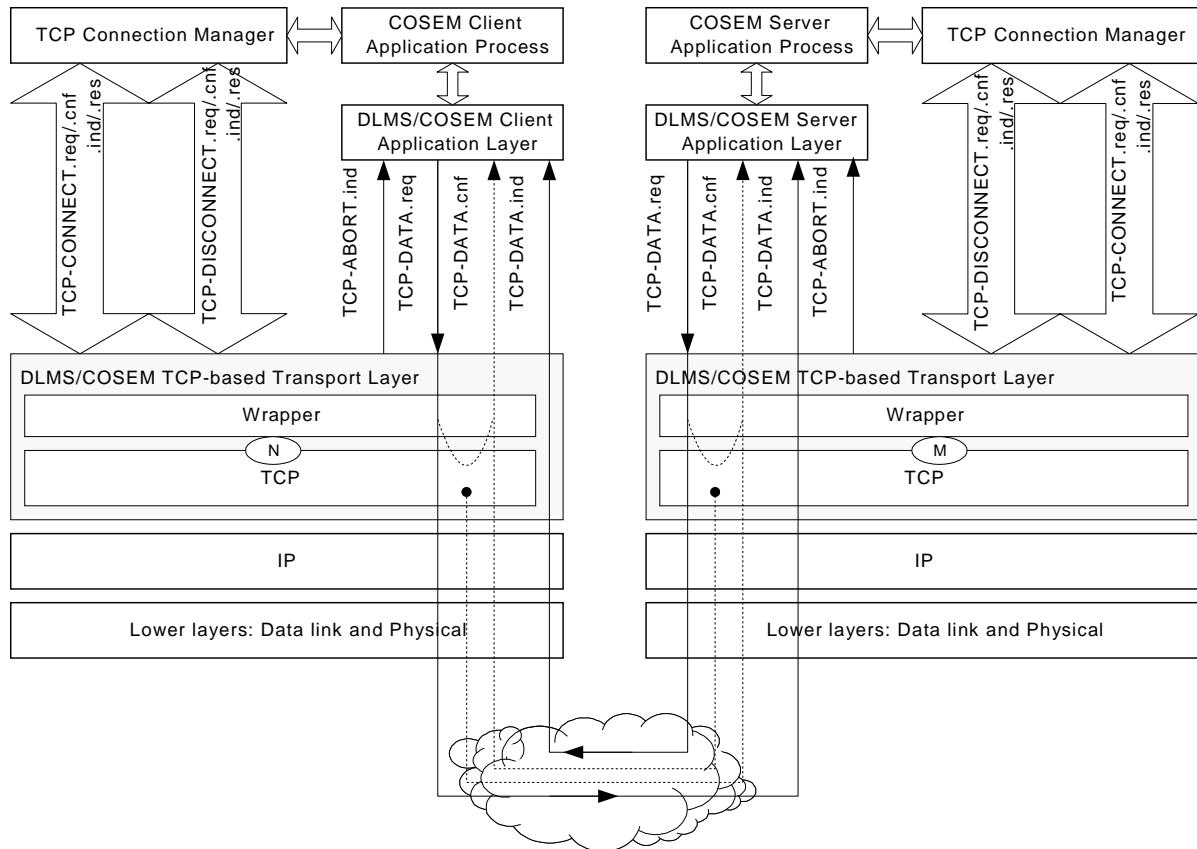


Figure 31 – Services of the DLMS/COSEM connection-oriented, TCP-based transport layer

7.4.2.2 The TCP-CONNECT service

7.4.2.2.1 TCP-CONNECT.request

Function

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

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
)
```

The Local_TCP_Port and Remote_TCP_Port parameters 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.

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.

7.4.2.2.2 TCP-CONNECT.indication

Function

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

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
)
```

The Local_TCP_Port and Remote_TCP_Port parameters indicate the two TCP ports between which the requested TCP connection is to be established. The Local_IP_Address and Remote_IP_Address parameters indicate the IP addresses of the two devices participating in the TCP connection.

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.

7.4.2.2.3 TCP-CONNECT.response

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

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

The Local_TCP_Port and Remote_TCP_Port parameters indicate the two TCP ports between which the connection is being established. The Local_IP_Address and Remote_IP_Address parameters indicate the IP addresses of the two physical devices participating in the TCP connection.

The Result parameter indicates that the service user TCP connection manager has accepted the requested TCP connection. Its value is always SUCCESS.

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.

7.4.2.2.4 TCP-CONNECT.confirm

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

```
TCP-CONNECT.confirm (
```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	94/500
-----------------------	------------	------------------------	--------

```

Local_TCP_Port,
Remote_TCP_Port,
Local_IP_Address,
Remote_IP_Address,
Result,
Reason_of_Failure
)

```

The Local_TCP_Port and Remote_TCP_Port parameters indicate the two TCP ports between which the connection is being established. The Local_IP_Address and Remote_IP_Address parameters indicate the IP addresses of the two physical devices participating in this TCP connection.

The Result parameter 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.

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.

7.4.2.3 The TCP-DISCONNECT service

7.4.2.3.1 TCP-DISCONNECT.request

Function

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

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
)

```

The service parameters are the identifiers of the TCP connection to be released. The Local_TCP_Port and Local_IP_Address parameters designate the local TCP port and IP address of the requesting device and application, the Remote_IP_Address and Remote_TCP_Port parameters refer to the remote device and application.

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.

7.4.2.3.2 TCP-DISCONNECT.indication

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

```

TCP-DISCONNECT.indication      (

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	95/500
-----------------------	------------	------------------------	--------

```

Local_TCP_Port,
Remote_TCP_Port,
Local_IP_Address,
Remote_IP_Address,
Reason
)

```

The Local_TCP_Port, Remote_TCP_Port, Local_IP_Address, Remote_IP_Address parameters 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.

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).

7.4.2.3.3 TCP-DISCONNECT.response

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

```

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

```

The Local_TCP_Port and Remote_TCP_Port parameters identify the two TCP ports between which the TCP connection has to be disconnected. The Local_IP_Address and Remote_IP_Address parameters indicate the IP addresses of the two physical devices participating in the TCP connection to be disconnected.

The Result parameter 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.

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).

7.4.2.3.4 TCP-DISCONNECT.confirm

Function

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	96/500
-----------------------	------------	------------------------	--------

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

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
)
```

The Local_TCP_Port and Remote_TCP_Port parameters identify the two TCP ports between which the TCP connection has to be disconnected. The Local_IP_Address and Remote_IP_Address parameters indicate the IP addresses of the two physical devices participating in the TCP connection to be disconnected.

The Result parameter 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.

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.

7.4.2.4 The TCP-ABORT service

7.4.2.4.1 TCP-ABORT.indication

Function

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

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
)
```

The Local_TCP_Port and Remote_TCP_Port parameters identify the two TCP ports the connection between which has aborted. The Local_IP_Address and Remote_IP_Address parameters indicate the IP addresses of the two physical devices having been participated in the TCP connection aborted.

The Reason parameter indicates the reason of the TCP abort. This parameter is optional.

Use

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	97/500
-----------------------	------------	------------------------	--------

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.

7.4.2.5 The TCP-DATA service

7.4.2.5.1 TCP-DATA.request

Function

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

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
)
```

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.

The Data_Length parameter indicates the length of the Data parameter in bytes.

The Data parameter contains the xDLMS APDU to be transferred to the peer AL.

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 7.3.3.2, as DATA. The TCP sublayer shall transmit the WPDU to the peer TCP sublayer as described in STD0007.

7.4.2.5.2 TCP-DATA.indication

Function

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

Semantics of the service primitive

The primitive shall provide parameters as follows:

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	98/500
-----------------------	------------	------------------------	--------

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

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.

The Data_Length parameter indicates the length of the Data parameter in bytes.

The Data parameter contains the xDLMS APDU received from the peer AL.

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.

7.4.2.5.3 TCP-DATA.confirm

Function

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

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
)
```

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.

The Confirmation_Type parameter indicates whether the confirmation service is a LOCAL or a REMOTE confirmation.

The value of the Result parameter 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 7.4.3.5.4.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	99/500
-----------------------	------------	------------------------	--------

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.

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

7.4.3.1 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 7.5.4) 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.

7.4.3.2 The wrapper protocol data unit (WPDU)

The wrapper protocol data unit is as it is specified at 7.3.3.2.

7.4.3.3 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.

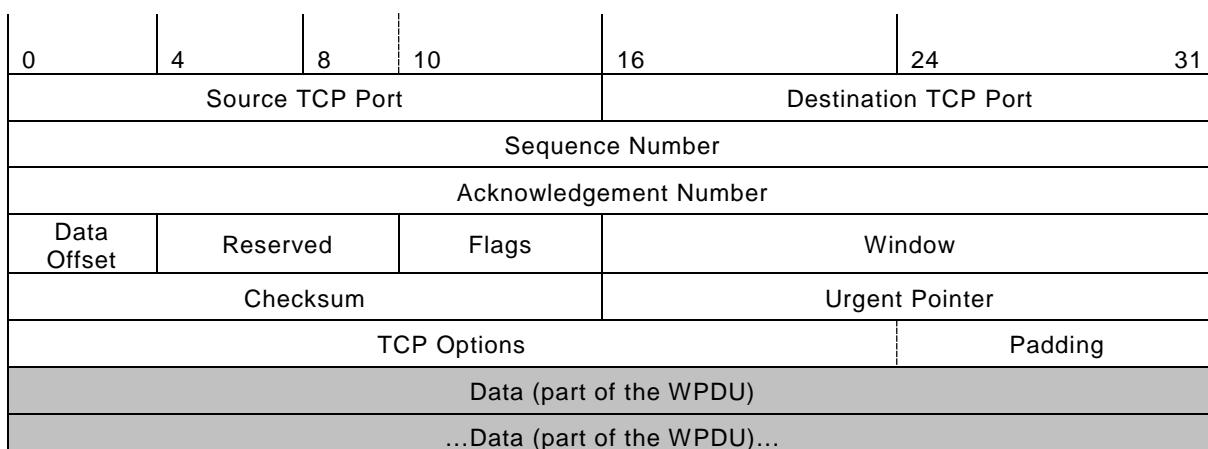


Figure 32 – The TCP packet format

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).

7.4.3.4 Reserved wrapper port numbers

Reserved wPort Numbers are specified in Table 2.

7.4.3.5 Definition of the procedures

7.4.3.5.1 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.

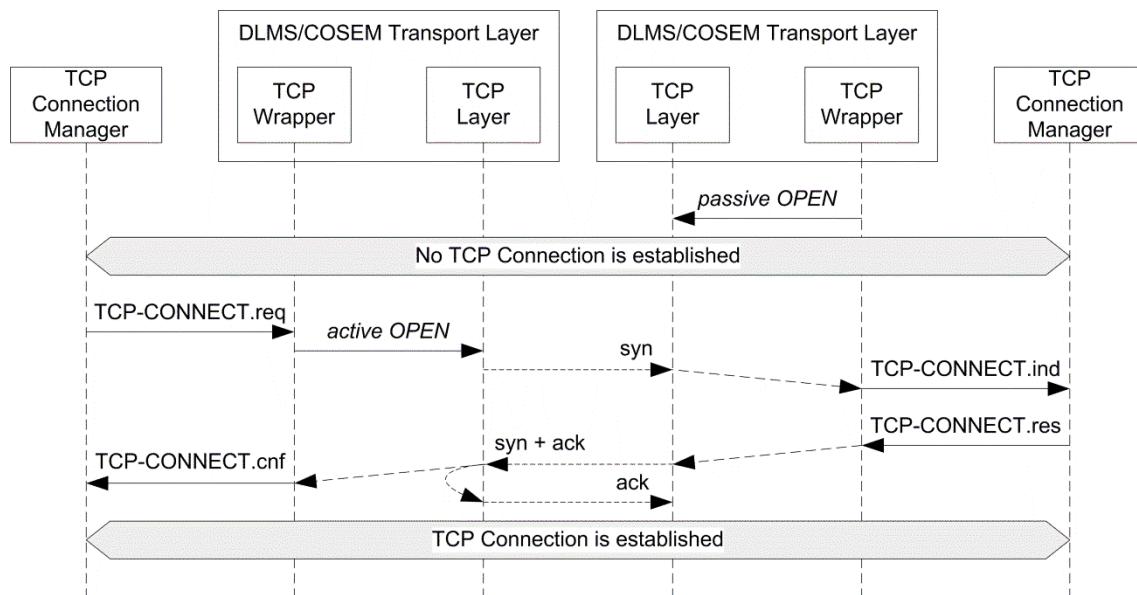


Figure 33 – TCP connection establishment

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 7.5.1.

7.4.3.5.2 TCP disconnection

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

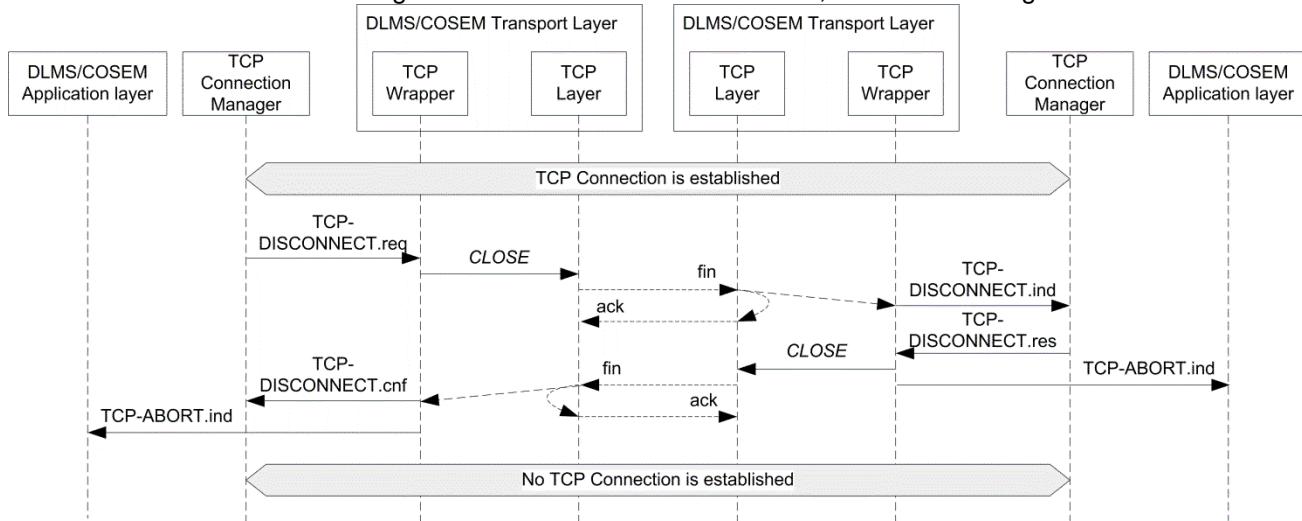


Figure 34 – TCP disconnection

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 7.5.2.

7.4.3.5.3 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.

7.4.3.5.4 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.

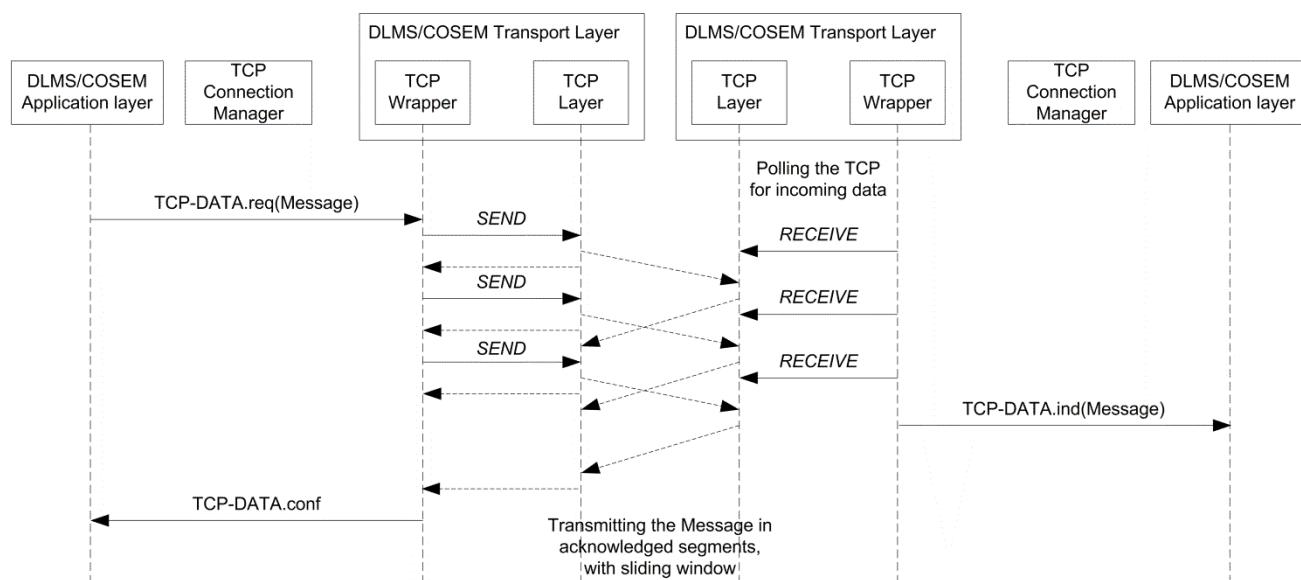


Figure 35 – Data transfer using the COSEM TCP-based transport layer

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 7.5.4.

7.4.3.5.5 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 7.5.

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

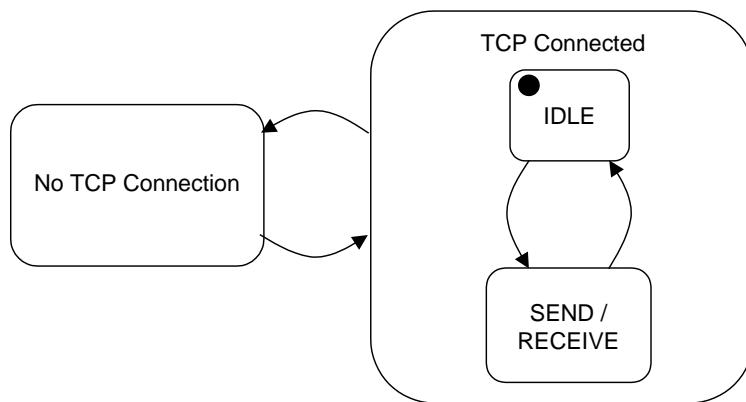


Figure 36 – High-level state transition diagram for the wrapper sublayer

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

7.5.1 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.

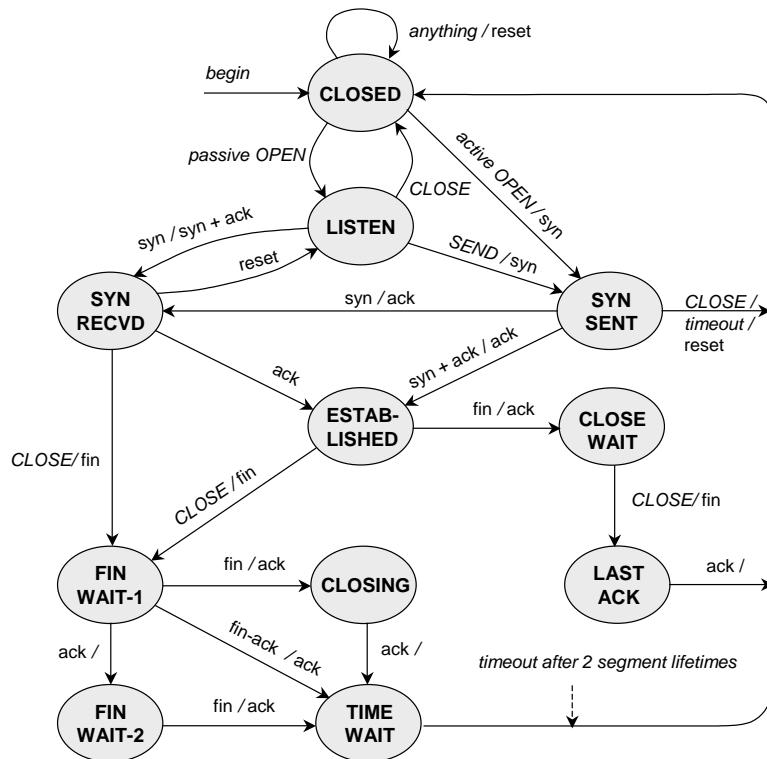
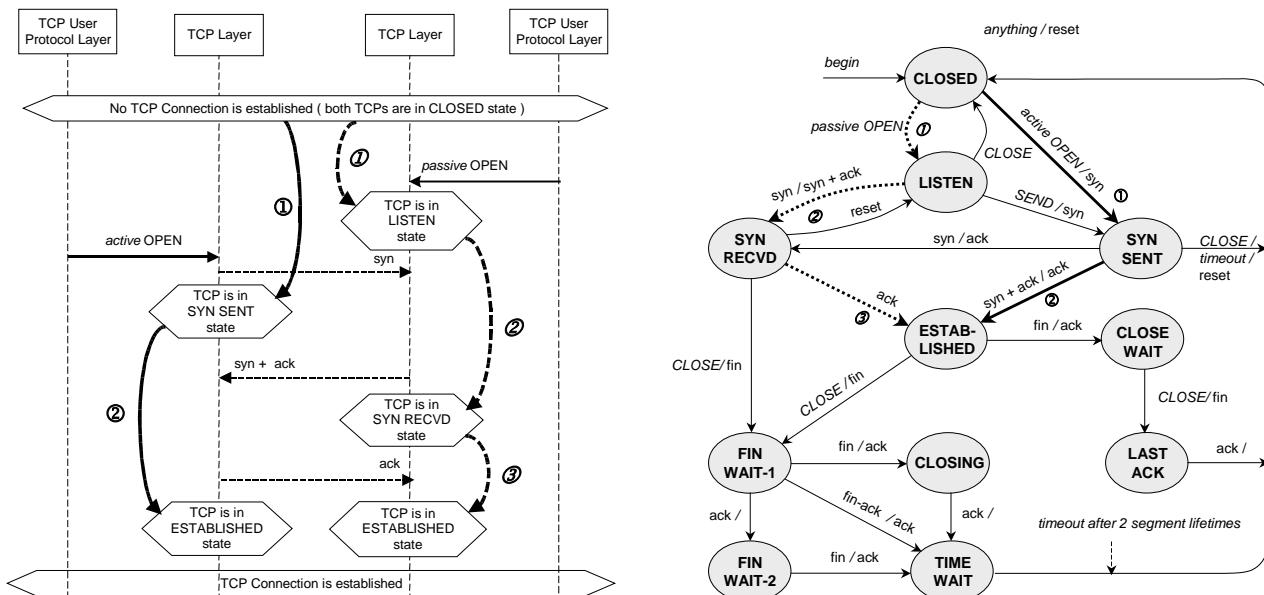


Figure 37 – TCP connection state diagram



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

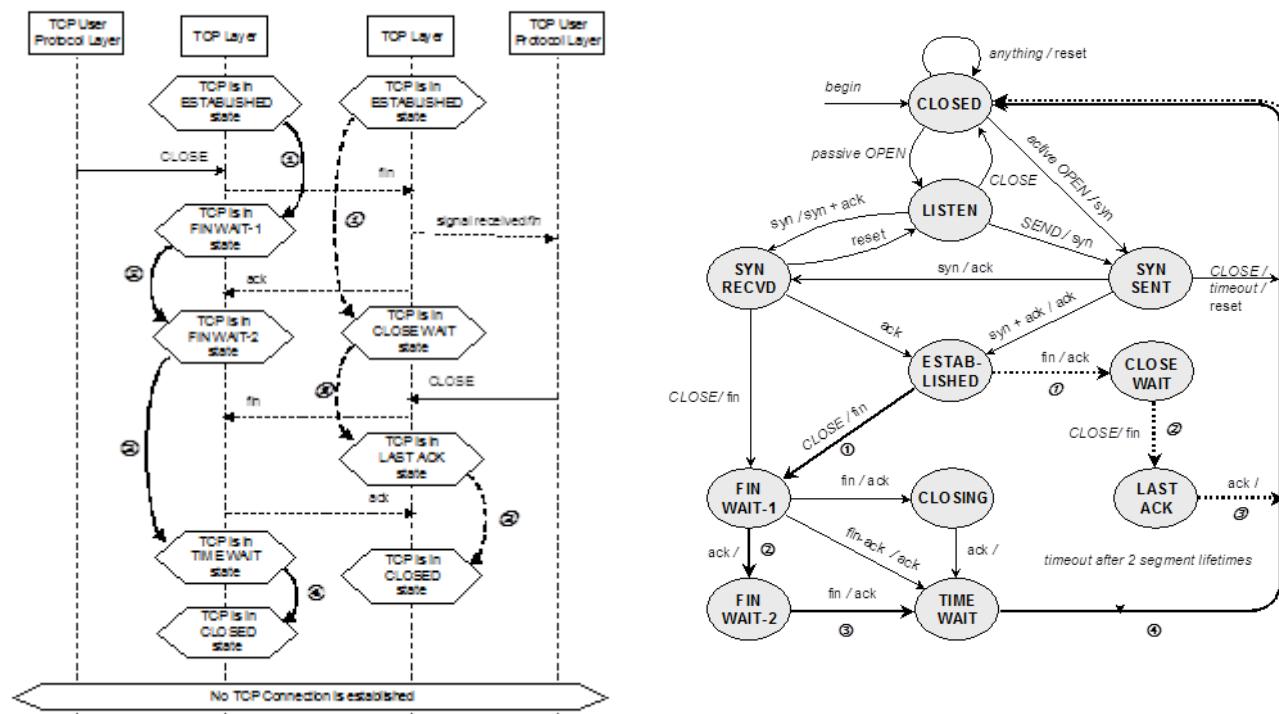
7.5.2 Closing a transport layer and a TCP connection

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

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 layer 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

7.5.3 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.

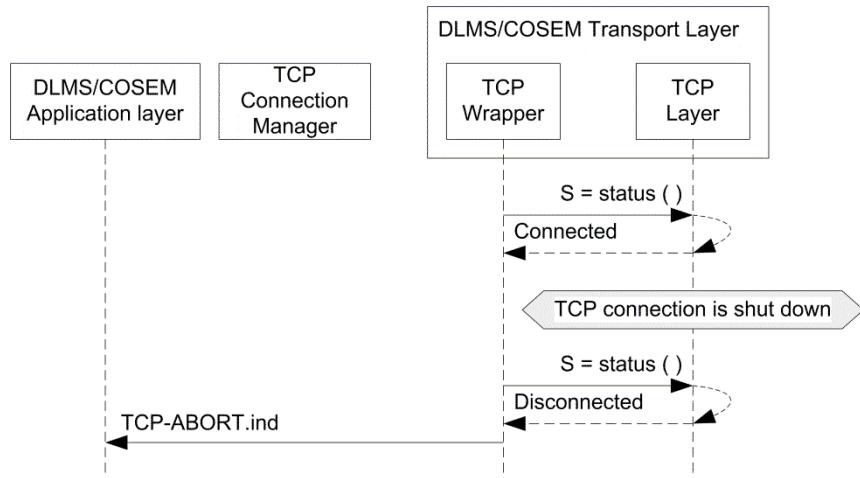


Figure 40 – Polling the TCP sublayer for TCP abort indication

7.5.4 Data transfer using the TCP-DATA service

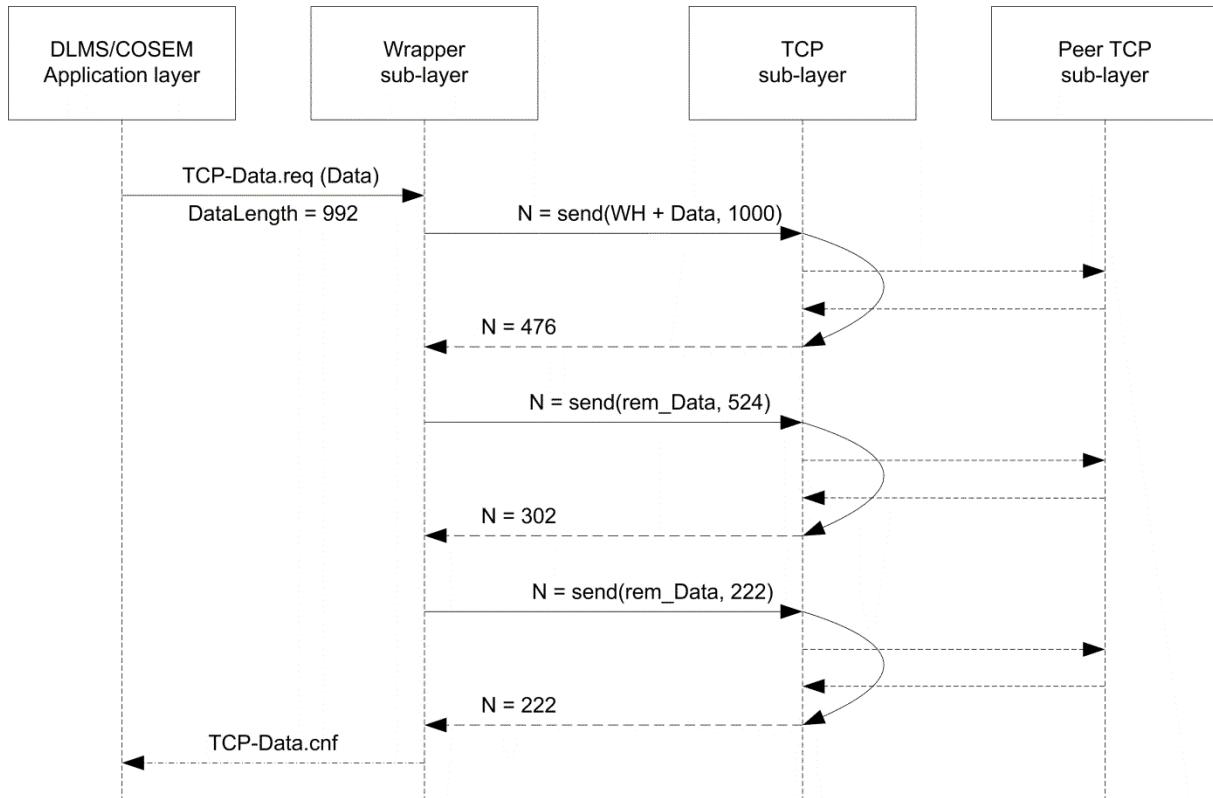
To send an APDU to the peer, the DLMS/COSEM AL simply invokes the TCP-DATA.request primitive of the DLMS/COSEM TCP-based TL. Also, when a complete APDU is received, this is indicated to the DLMS/COSEM AL with the help of the TCP-DATA.indication primitive. Thus, for the AL the TL behaves as if it would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

However, as TCP is a streaming protocol, not preserving data boundaries – as described in 7.4.3.1 – nothing ensures that an APDU is actually transmitted in one TCP packet. As already mentioned in 7.4.3.5.4, 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.

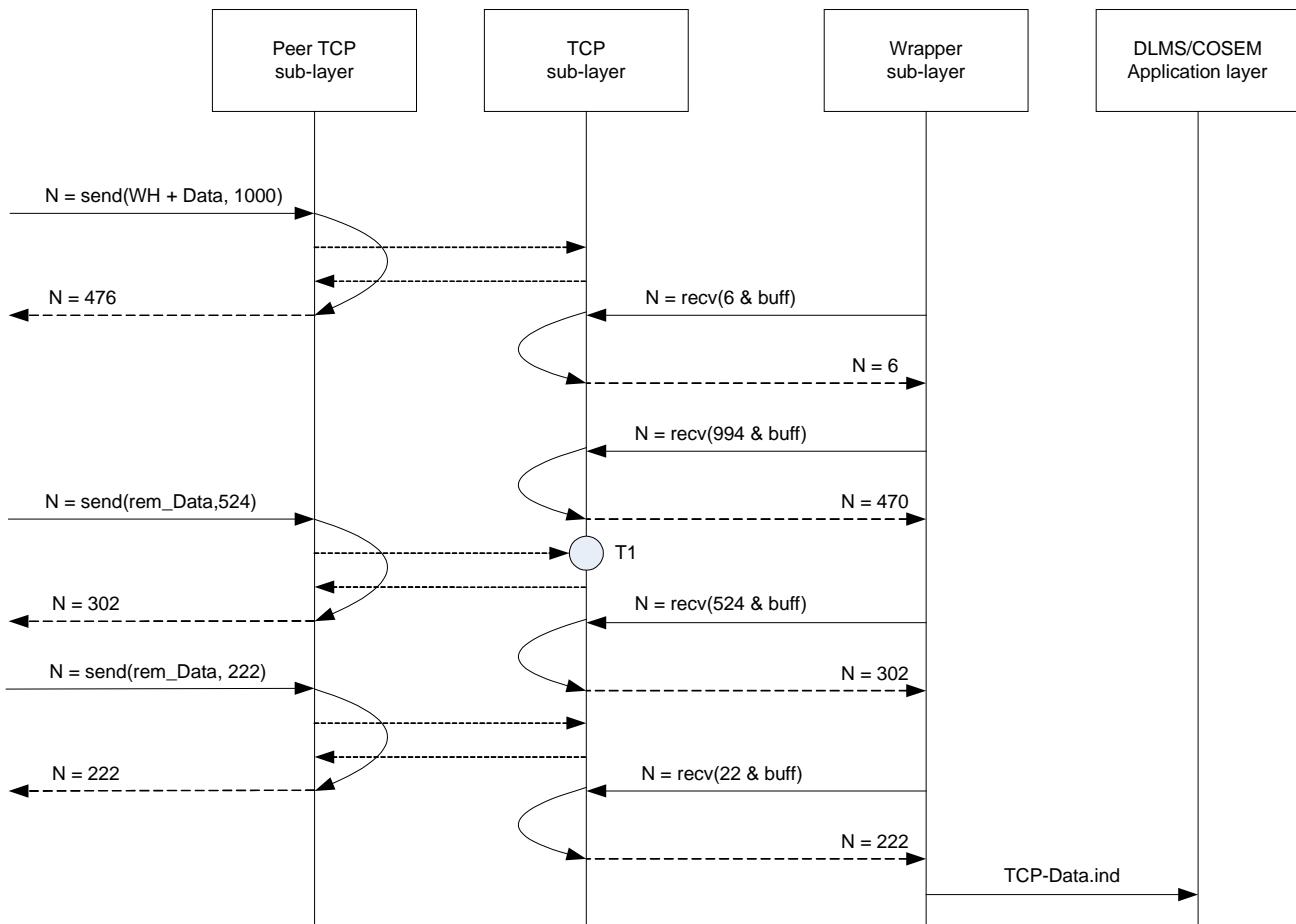
**Figure 41 – Sending an APDU in three TCP packets**

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 7.4.3.5.4, 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.

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 meter 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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	110/500
-----------------------	------------	------------------------	---------

link level error conditions, by sending command and supervisory frames. The secondary station(s) respond(s) by sending response frames.

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.

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 43 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.

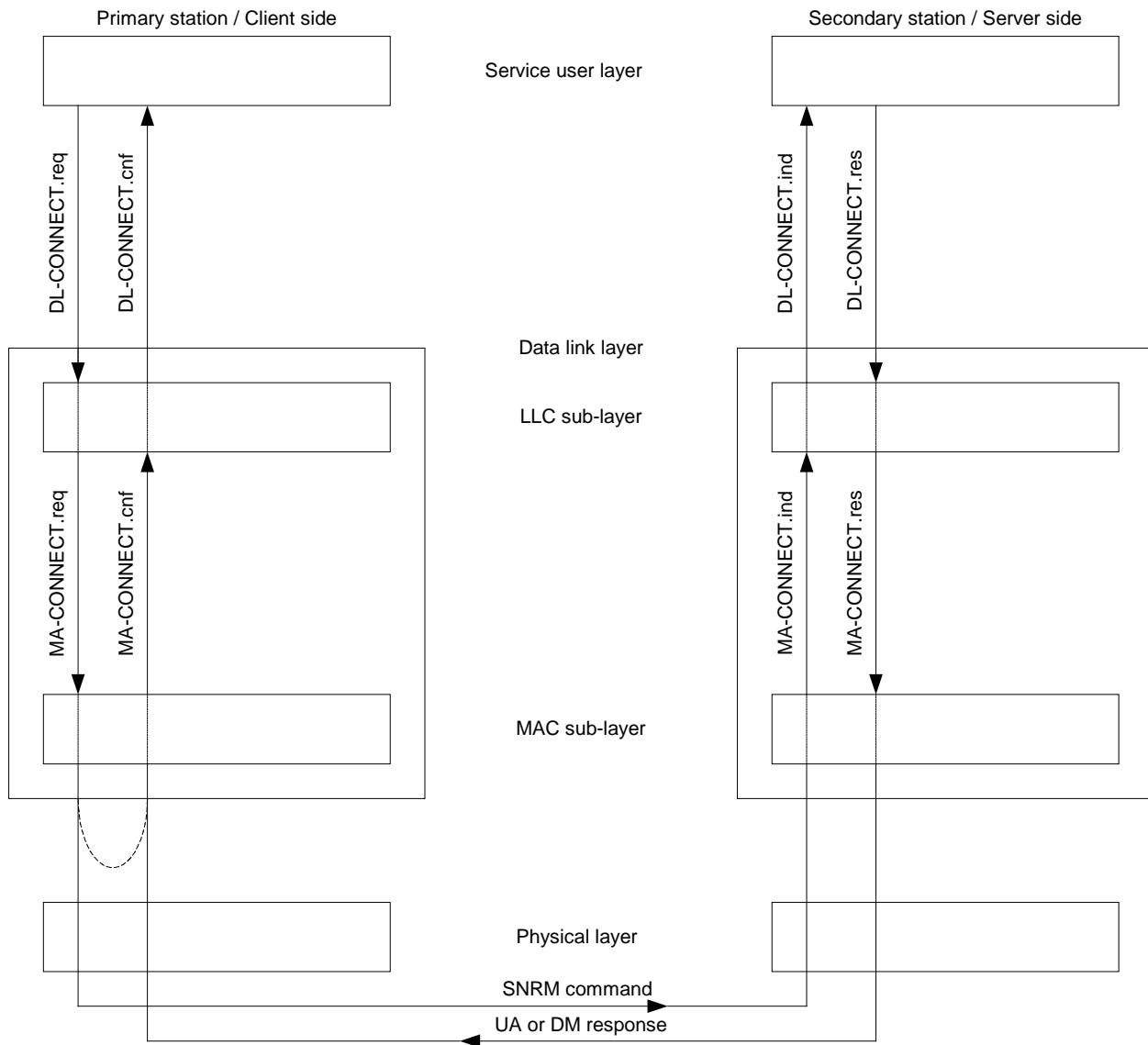


Figure 43 – Data link layer services for data link connection

8.2.2.2 DL-CONNECT.request and MA-CONNECT.request

Function

These primitives are the service request primitives for the connection establishment service.

Semantics of the service primitives

The primitives shall provide parameters as follows:

DL-CONNECT.request (Destination_MSAP ⁵ , Source_MSAP, User_Information)	MA-CONNECT.request (Destination_MSAP, Source_MSAP, User_Information)
---	---

The Destination_MSAP and Source_MSAP parameters identify the referenced data link layer connection to be set up.

⁵ MSAP in this environment is equal to the HDLC address.

The User_Information parameter is optional. The specification of its contents is not within the Scope of this Technical Report.

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

Function

These primitives are the service indication primitives for the connection establishment service.

Semantics of the service primitives

The primitives shall provide parameters as follows:

DL-CONNECT.indication (Destination_MSAP, Source_MSAP, User_Information)	MA-CONNECT.indication (Destination_MSAP, Source_MSAP, User_Information)
--	--

The Destination_MSAP and Source_MSAP parameters identify the referenced data link layer connection to be set up.

The User_Information parameter is optional. The specification of its contents is not within the Scope of this Technical Report.

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

Function

These primitives are the service response primitives for the connection establishment service.

Semantics of the service primitives

The primitives shall provide parameters as follows:

```

DL-CONNECT.response (
    Destination_MSAP,
    Source_MSAP,
    Result,
    User_Information
)
MA-CONNECT.response (
    Destination_MSAP,
    Source_MSAP,
    Result,
    User_Information
)

```

The Destination_MSAP and Source_MSAP parameters identify the referenced data link layer connection being set up.

The Result parameter 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).

The User_Information parameter is optional. It may be present only when Result == NOK. The specification of its contents is not within the Scope of this Technical Report.

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

Function

These primitives are the service .confirm primitives for the connection establishment service.

Semantics of the service primitives

The primitives shall provide parameters as follows:

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	115/500
-----------------------	------------	------------------------	---------

```

DL-CONNECT.confirm      (
    Destination_MSAP,
    Source_MSAP,
    Result,
    User_Information
)
MA-CONNECT.confirm      (
    Destination_MSAP,
    Source_MSAP,
    Result,
    User_Information
)

```

The Destination_MSAP and Source_MSAP parameters reference data link layer connection, which is confirmed by the service primitive.

The Result parameter 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.

The User_Information parameter 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.

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

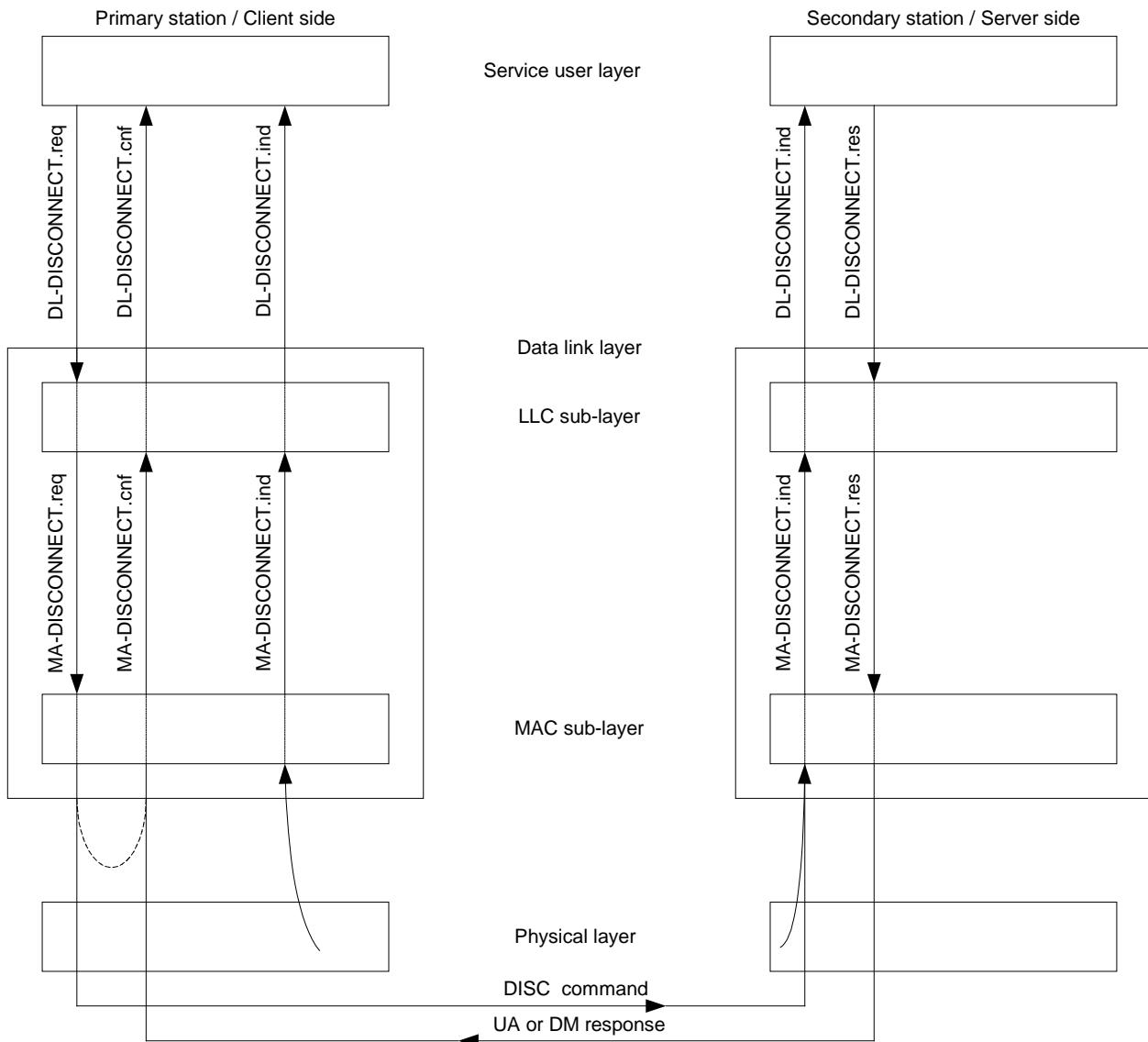
Figure 44 shows the services required of the primary station (client side) and secondary station (server side) data link layers by the service user layer for data link connection termination.

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 44 – Data link layer services for data link disconnection**

8.2.3.2 DL-DISCONNECT.request and MA-DISCONNECT.request

Function

These primitives are the service request primitives for the connection termination service.

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
)
```

The Destination_MSAP and Source_MSAP parameters identify the data link layer connection to be terminated.

The User_Information parameter is optional. The specification of its contents is not within the Scope of this Technical Report.

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

Function

These primitives are the service indication primitives for the connection termination service.

Semantics of the service primitive

The primitives shall provide parameters as follows:

DL-DISCONNECT.indication (Destination_MSAP, Source_MSAP, Reason, Unnumbered_Send_Status, User_Information)	MA-DISCONNECT.indication (Destination_MSAP, Source_MSAP, Reason, Unnumbered_Send_Status, User_Information)
---	---

The Destination_MSAP and Source_MSAP parameters identify the data link layer connection to be terminated.

The Reason parameter 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.

The User_Information parameter 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.

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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	118/500
-----------------------	------------	------------------------	---------

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

Function

These primitives are the service response primitives for the connection termination service.

Semantics of the service primitives

The primitives shall provide parameters as follows:

DL-DISCONNECT.response (Destination_MSAP, Source_MSAP, Result)	MA-DISCONNECT.response (Destination_MSAP, Source_MSAP, Result)
---	---

The Destination_MSAP and Source_MSAP parameters identify the data link layer connection being terminated.

The Result parameter 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.

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

Function

These primitives are the service .confirm primitives for the connection termination service.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	119/500
-----------------------	------------	------------------------	---------

Semantics of the service primitives

The primitives shall provide parameters as follows:

DL-DISCONNECT.confirm (Destination_MSAP, Source_MSAP, Result)	MA-DISCONNECT.confirm (Destination_MSAP, Source_MSAP, Result)
--	--

The Destination_MSAP and Source_MSAP parameters identify the data link layer connection, the termination of which is confirmed by the service primitive.

The Result parameter 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.

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 45 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.

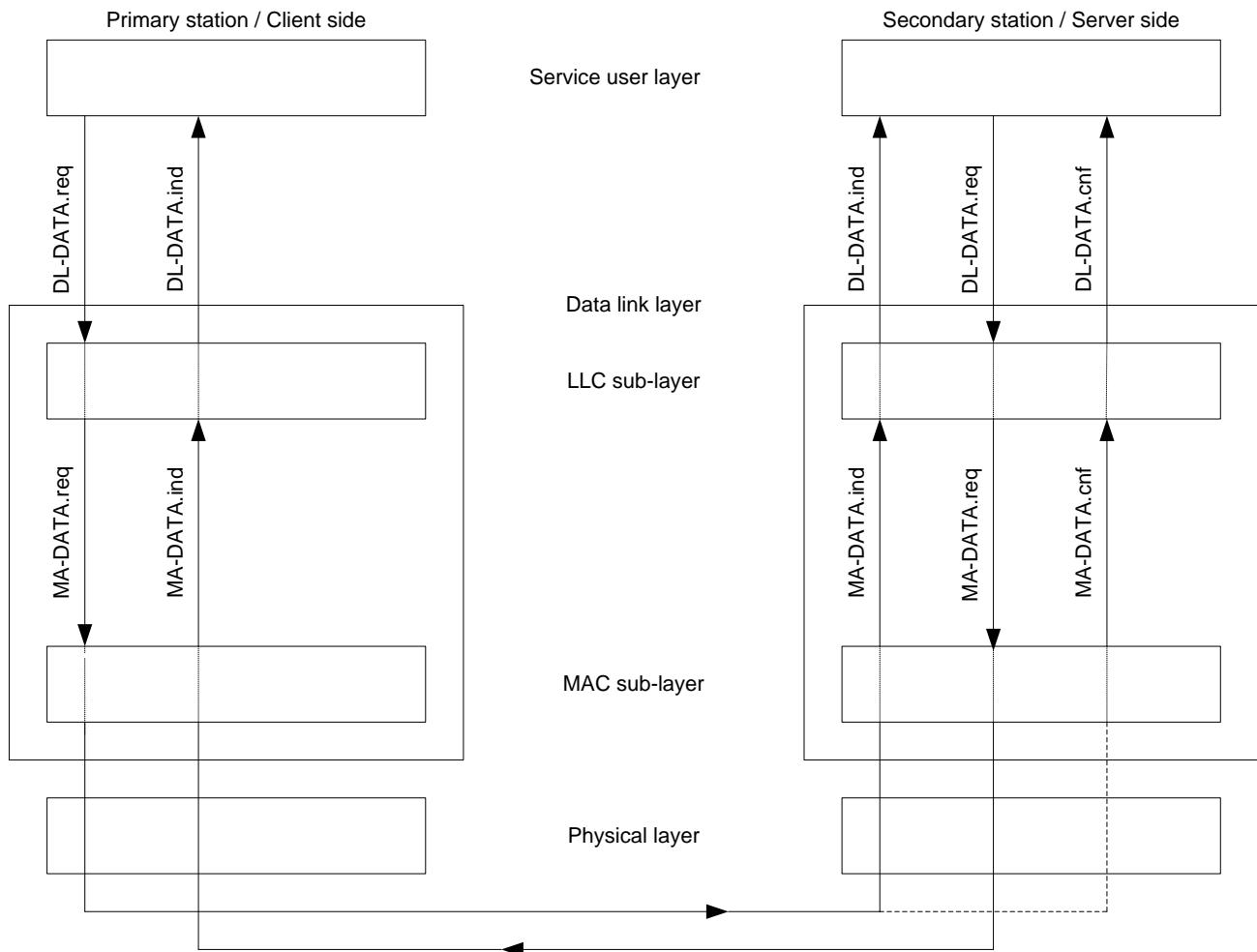


Figure 45 – Data link layer data transfer services

8.2.4.2 DL-DATA.request and MA-DATA.request

Function

These primitives are the service request primitives for the data transfer service.

Semantics of the service primitives

The primitives shall provide parameters as follows:

DL-DATA.request (MA-DATA.request (
Destination_LSAP,	Destination_MSAP,
Source_LSAP,	Source_MSAP,
LLC_Quality,	Frame_type,
Destination_MSAP,	Data
Source_MSAP,)
Frame_type,	
Data	
)	

The Destination_LSAP and Source_LSAP parameters 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.

The Destination_MSAP and Source_MSAP parameters 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.

The Frame_Type parameter 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.

The Data parameter 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).

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

Function

These primitives are the service indication primitives for the data transfer service.

Semantics of the service primitives

The primitive shall provide parameters as follows:

DL-DATA.indication	(MA-DATA.indication	(
Destination_LSAP,		Destination_MSAP,	
Source_LSAP,		Source_MSAP,	
LLC_Quality,		Frame_type,	
Destination_MSAP,		Data	
Source_MSAP,)
Frame_type,			
Data)		

The Destination_LSAP and Source_LSAP parameters 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.

The Destination_MSAP and Source_MSAP parameters 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.

The Frame_Type parameter 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.

The Data parameter 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).

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.

8.2.4.4 DL-DATA.confirm and MA-DATA.confirm

Function

These primitives are the service .confirm primitives for the data transfer service.

Semantics of the service primitives

The primitives shall provide parameters as follows:

DL-DATA.confirm (Destination_Lsap, Source_Lsap, LLC_Quality, Destination_Msap, Source_Msap, Frame_type, Result)	MA-DATA.confirm (Destination_Msap, Source_Msap, Frame_type, Result)
--	---

The Destination_Lsap and Source_Lsap parameters identify the data link layer connection referenced.

The Destination_Msap and Source_Msap parameters identify the local and remote MSAP-s involved in the data unit transmission. The Destination_Msap shall be an individual address.

The Frame_Type parameter indicates the type of the HDLC frame which is confirmed. Valid frame types are I-FIRST-FRAGMENT, I-FRAGMENT and I-LAST-FRAGMENT.

The value of the Result parameter indicates the result of the transmission of the received MSDU resp. LSDU. Possible values are OK and NOK.

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

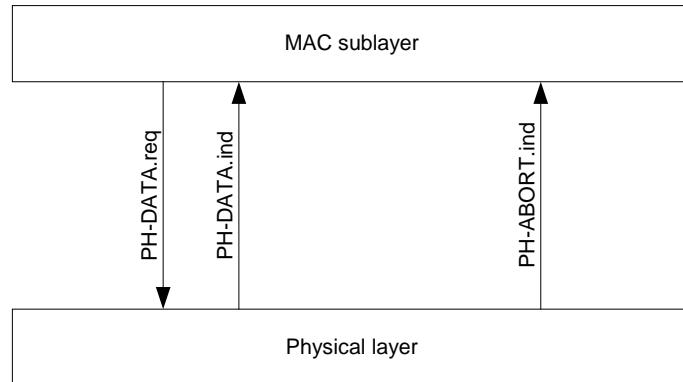
8.2.5.1 Overview

Figure 46 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	123/500
-----------------------	------------	------------------------	---------

**Figure 46 – 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 47.

Destination (remote) LSAP	Source (local) LSAP	Control	Information
8 bits	8 bits	8 or 16 bits	n*8 bits

Figure 47 – The ISO/IEC 8802-2 LLC PDU format

For the purposes of DLMS/COSEM, this LLC PDU format is used as shown on Figure 48:

Destination (remote) LSAP	Source (local) LSAP	LLC_Quality	Information
8 bits: 0xE6	8 bits: 0xE6 or 0xE7	8 bits: 0x00	n*8 bits

Figure 48 – 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 3 and in Table 4 respectively.

Table 3 – 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 4 – 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

⁶⁾ 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 49:

Flag	Frame format	Dest. address	Src. address	Control	HCS	Information	FCS	Flag
------	--------------	---------------	--------------	---------	-----	-------------	-----	------

Figure 49 – 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 50.

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 50 – 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 51:

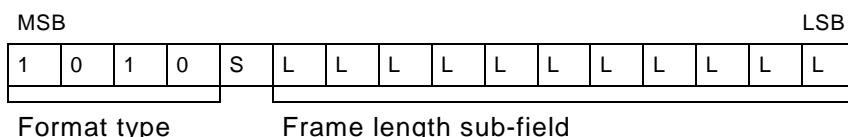


Figure 51 – The frame format field

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.

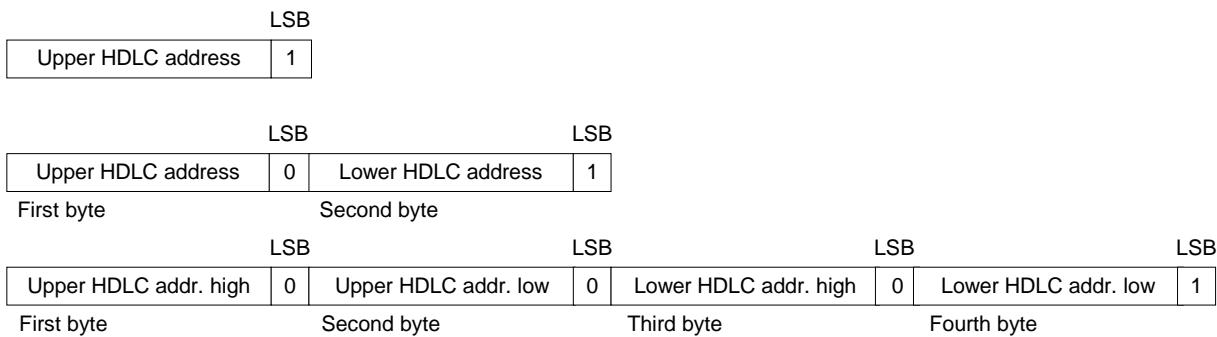
The client address shall always be expressed on one byte.

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 52. 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 52 – Valid server address structures****8.4.2.3 Reserved special HDLC addresses**

The following special HDLC addresses are reserved:

Table 5 – Table of reserved client addresses

Reserved HDLC addresses	
No-station	0x00
Client Management Process	0x01
Public Client	0x10
<i>Open for client SAP assignment</i>	0x02 ... 0x0F 0x11 ... 0xFF

Table 6 – 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
<i>Open for server SAP assignment</i>	0x10...0x7E	0x0010...0x3FFE
All-station (Broadcast)	0x7F	0x3FFF

Reserved lower HDLC addresses		
	One byte address	Two byte address
No-station	0x00	0x0000
Reserved for future use	0x01...0x0F	0x0001...0x000F
<i>Open for server SAP assignment</i>	0x10...0x7D	0x0010...0x3FFD
CALLING 7) Physical Device	0x7E	0x3FFE
All-station (Broadcast)	0x7F	0x3FFF

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_B

Server HDLC Address (using four bytes addressing)

lower HDLC Address = 0x3FFF = 0011111111111111_B All-station (Broadcast) Address
upper HDLC Address = 0x1234 = 0001001000110100_B

⁷ The meaning of the CALLING Physical Device is discussed in 8.4.2.4.

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	0	1	1
First byte		Second byte		Third byte		Fourth byte		Fifth byte	
Destination address								Source address	

Figure 53 – 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 7.

Table 7 – 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 8.

Table 8 – 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	131/500
-----------------------	------------	------------------------	---------

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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	132/500
-----------------------	------------	------------------------	---------

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	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	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 9.

Table 9 – 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 _D)	Max_info_field_length-receive (0x06) / 0x40 (64 _D)	Max_info_field_length-receive (0x06) / 0x40 (64 _D)
Max_info_field_length-receive (0x06) / 0x80 (128 _D)	Max_info_field_length-transmit (0x05) / 0x80 (128 _D)	Max_info_field_length-transmit (0x05) / 0x80 (128 _D)
Window size-transmit (0x07) / 0x0001 (1 _D)	Window size-receive (0x08) / 0x0007 (7 _D)	Window size-receive (0x08) / 0x0001 (1 _D)
Window size-receive (0x08) / 0x0007 (7 _D)	Window size-transmit (0x07) / 0x0007 (7 _D)	Window size-transmit (0x07) / 0x0007 (7 _D)
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 9.

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.

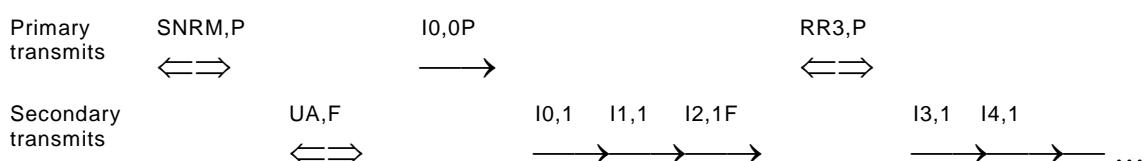
A time-out function shall be used in the primary station to monitor the responses of the secondary station.

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.

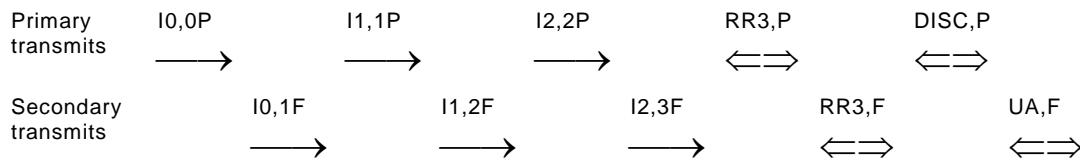
The following notation is used:

→ frame with info, ↔ frame without info, P / F poll/final bit, I / RR: Frame type, "n, n" values of N(S) and N(R)

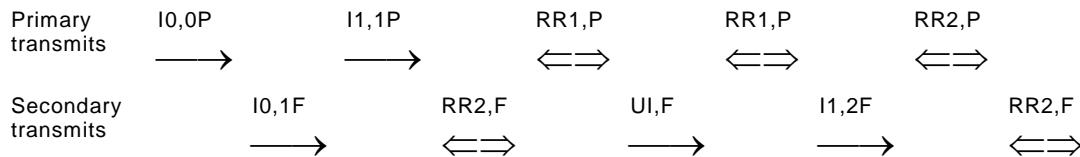
Example 1: Start-up procedure and information transfer without transmission errors (window size=3):



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 50).

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 54 shows the corresponding message sequence chart for the GET.request AL service. The Read.request service is treated in the same way.

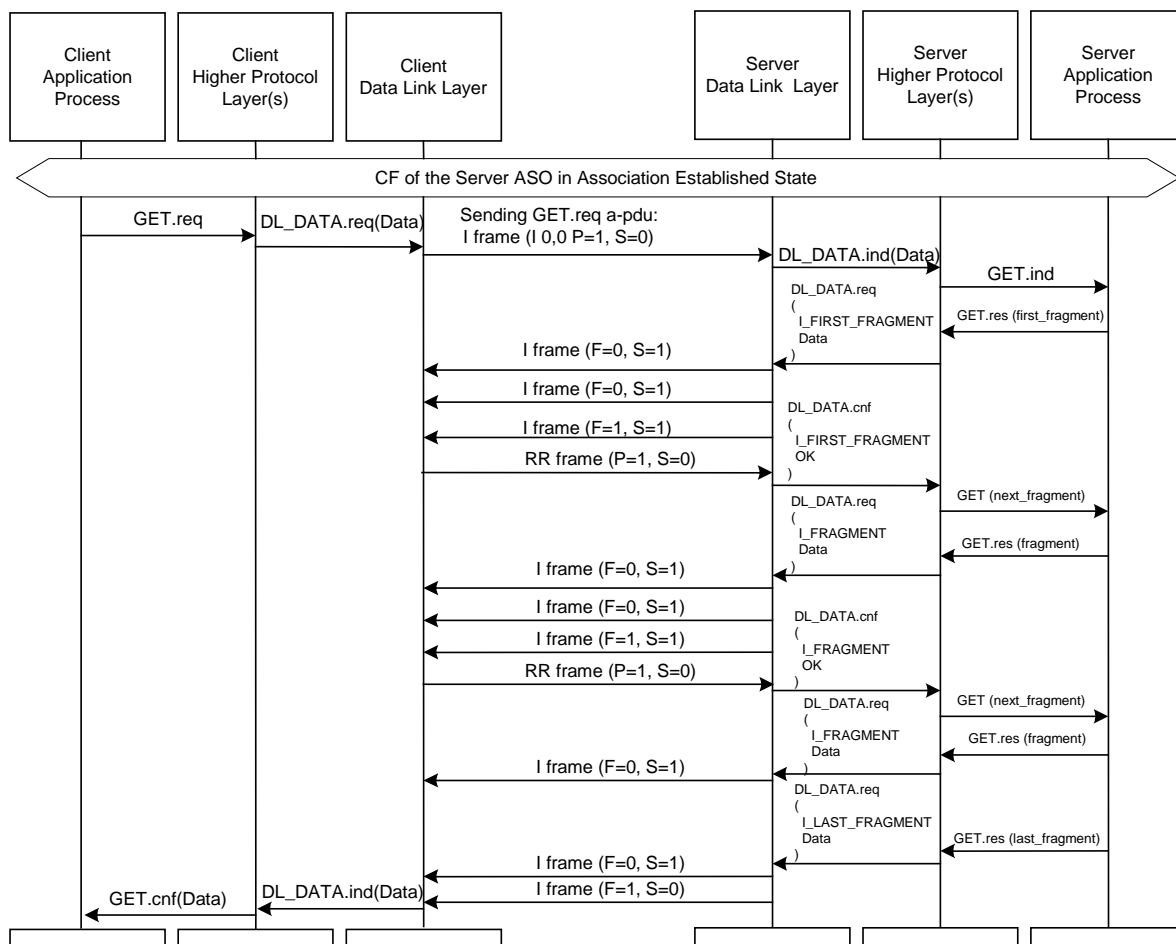


Figure 54 – MSC for long MSDU transfer in a transparent manner

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 55 shows an example with two physical devices, each of them including two logical devices.

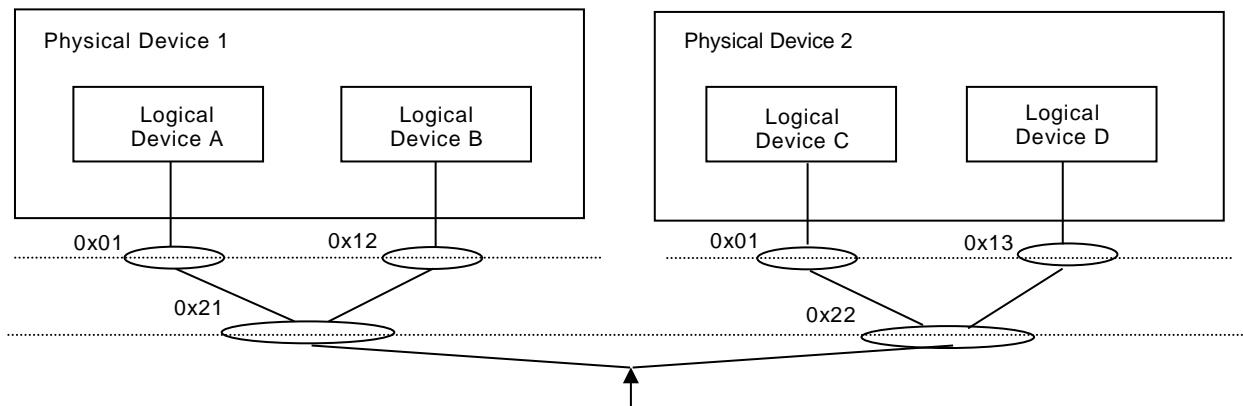


Figure 55 – Example configuration to illustrate broadcasting

As shown on Figure 55, the MAC addresses of these devices are the following:

Table 10 – Summary of MAC addresses for the example

	Upper MAC address (logical device)	Lower MAC address (physical device)
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 11.

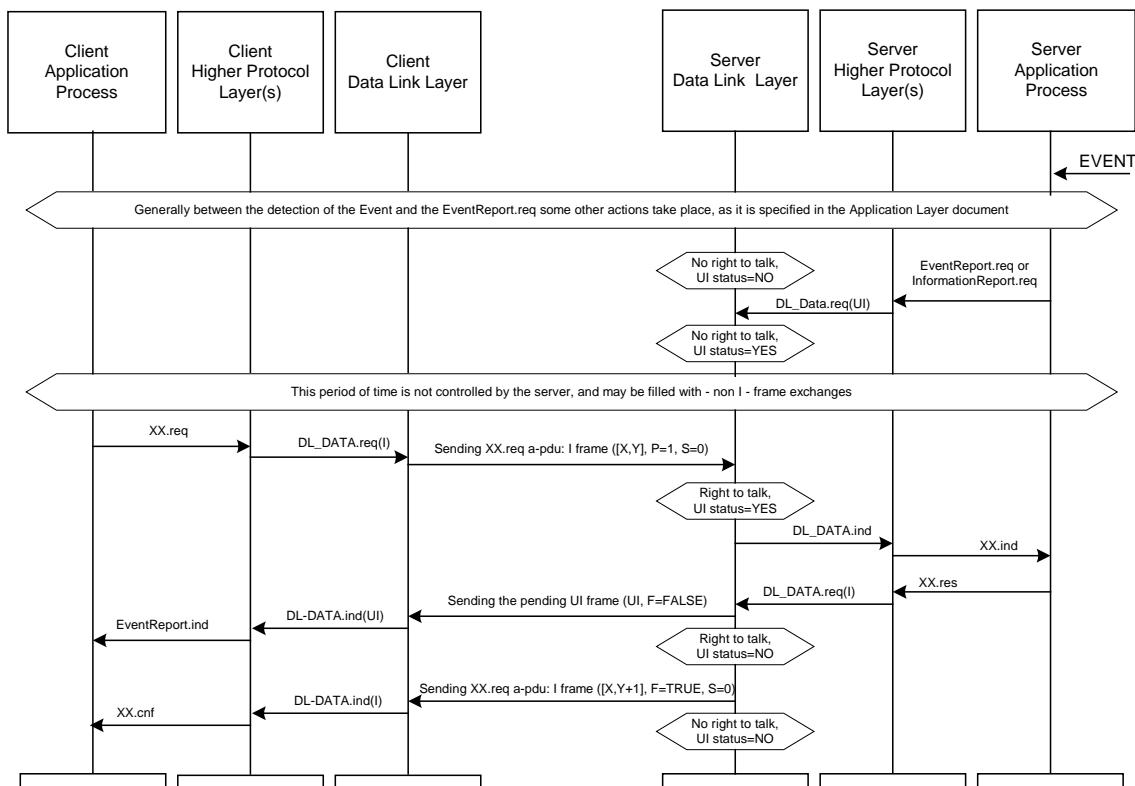
Table 11 – Broadcast UI frame handling

Lower MAC address (Physical device address)	Upper MAC address (Logical device address)	UI message handling
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 56 – Sending out a pending UI frame with a .response data**

Pending UI frames can be sent on the following conditions:

- a) 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 56;
- b) 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 57. For the use of the R frame, see 8.4.3.3;
- c) 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 58.

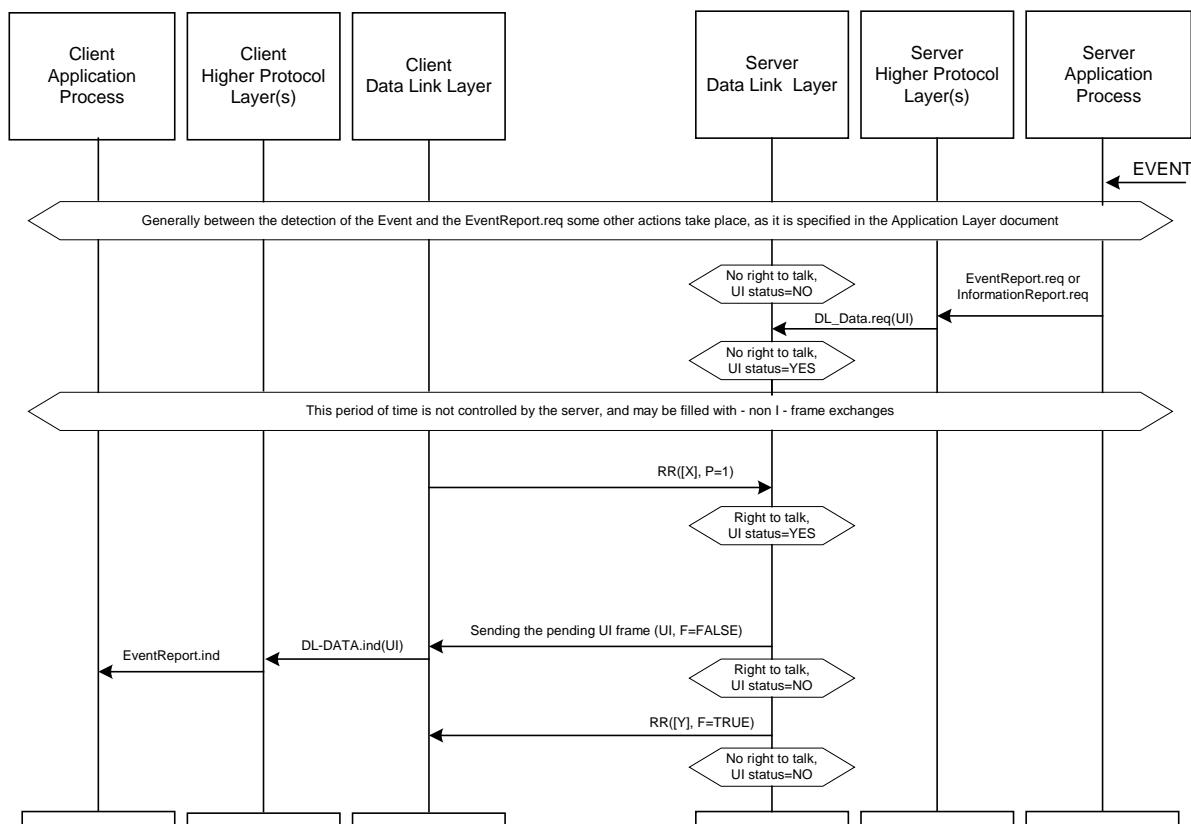


Figure 57 – Sending out a pending UI frame with a response to a RR frame

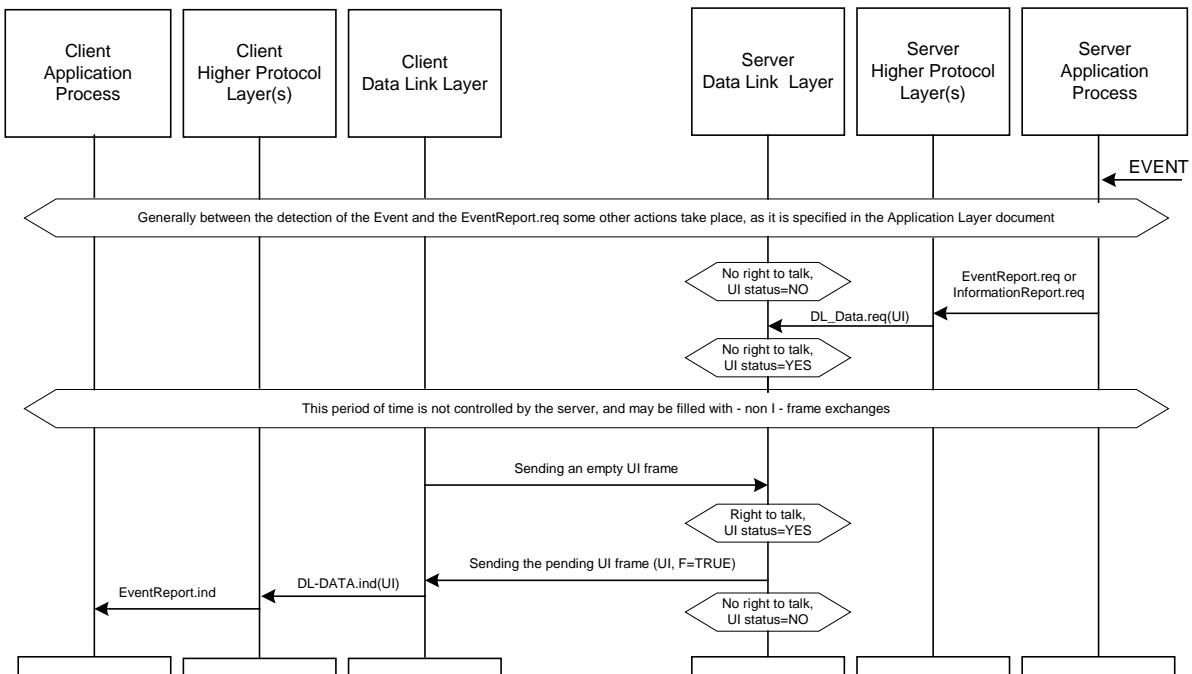


Figure 58 – Sending out a pending UI frame on receipt of an empty UI frame

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 meter 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 meter, 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 6).

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 retrial 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 MAC_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.

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 59 shows a simplified state transition diagram for the server MAC sublayer.

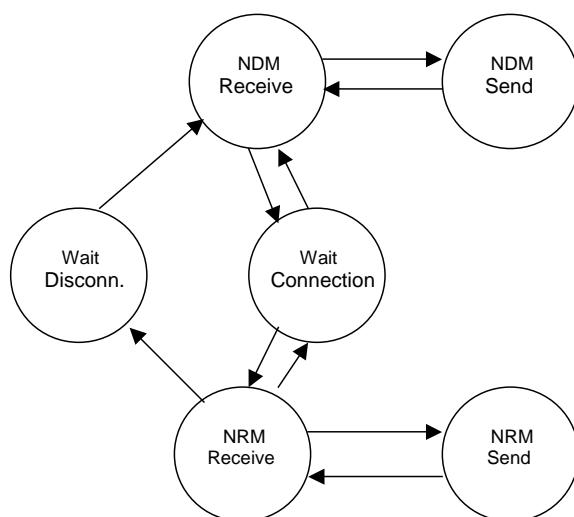


Figure 59 – State transition diagram for the server MAC sublayer

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.

V – first bit transmitted			last bit transmitted – V
0111 1110	1100 0000	1111 1100	1101 1010 0011 0111
flag	address	control	FCS

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,
    0xbdc9, 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, 0xfddc, 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,
0xffffcf, 0xee46, 0xdcd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
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, 0xa50, 0xb5d9, 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, 0xd68, 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.
 */

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	147/500
-----------------------	------------	------------------------	---------

```

* 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 60 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.

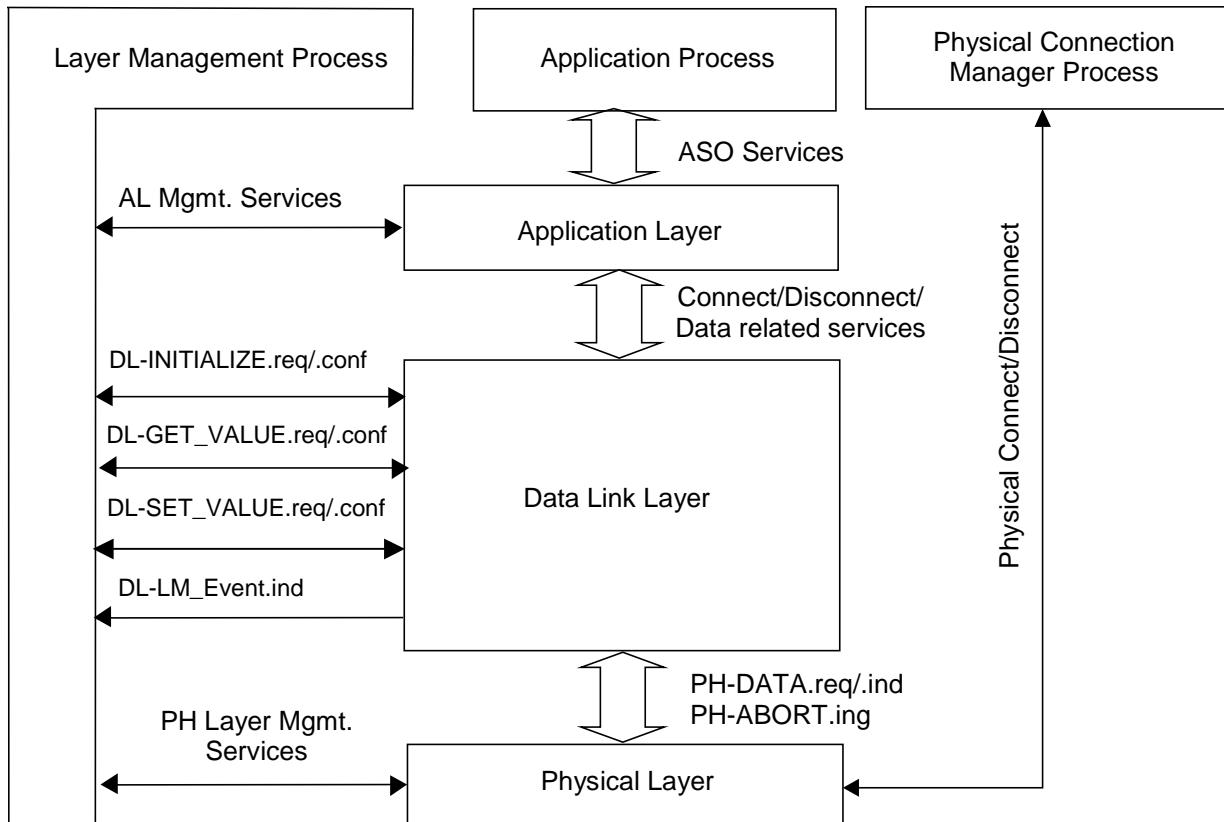


Figure 60 – Layer management services

8.6.2 Data link layer management service definitions

8.6.2.1 DL-INITIALIZE.request

Function

This primitive is the service request primitive of the DL-INITIALIZE service.

Semantics of the service primitive

The semantics of this service primitive is as follows:

DL-INITIALIZE.request
(
)

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

Function

This primitive is the service confirm primitive of the DL-INITIALIZE service.

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.

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*Function*

This primitive is the service request primitive of the DL-GET_VALUE service.

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.

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*Function*

This primitive is the service confirm primitive of the DL-GET_VALUE service.

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.

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*Function*

This primitive is the service request primitive of the DL-SET_VALUE service.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	150/500
-----------------------	------------	------------------------	---------

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.

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*Function*

This primitive is the service confirm primitive of the DL-SET_VALUE service.

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.

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*Function*

This primitive is the service indication primitive of the DL-LM_EVENT service.

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.

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 61.

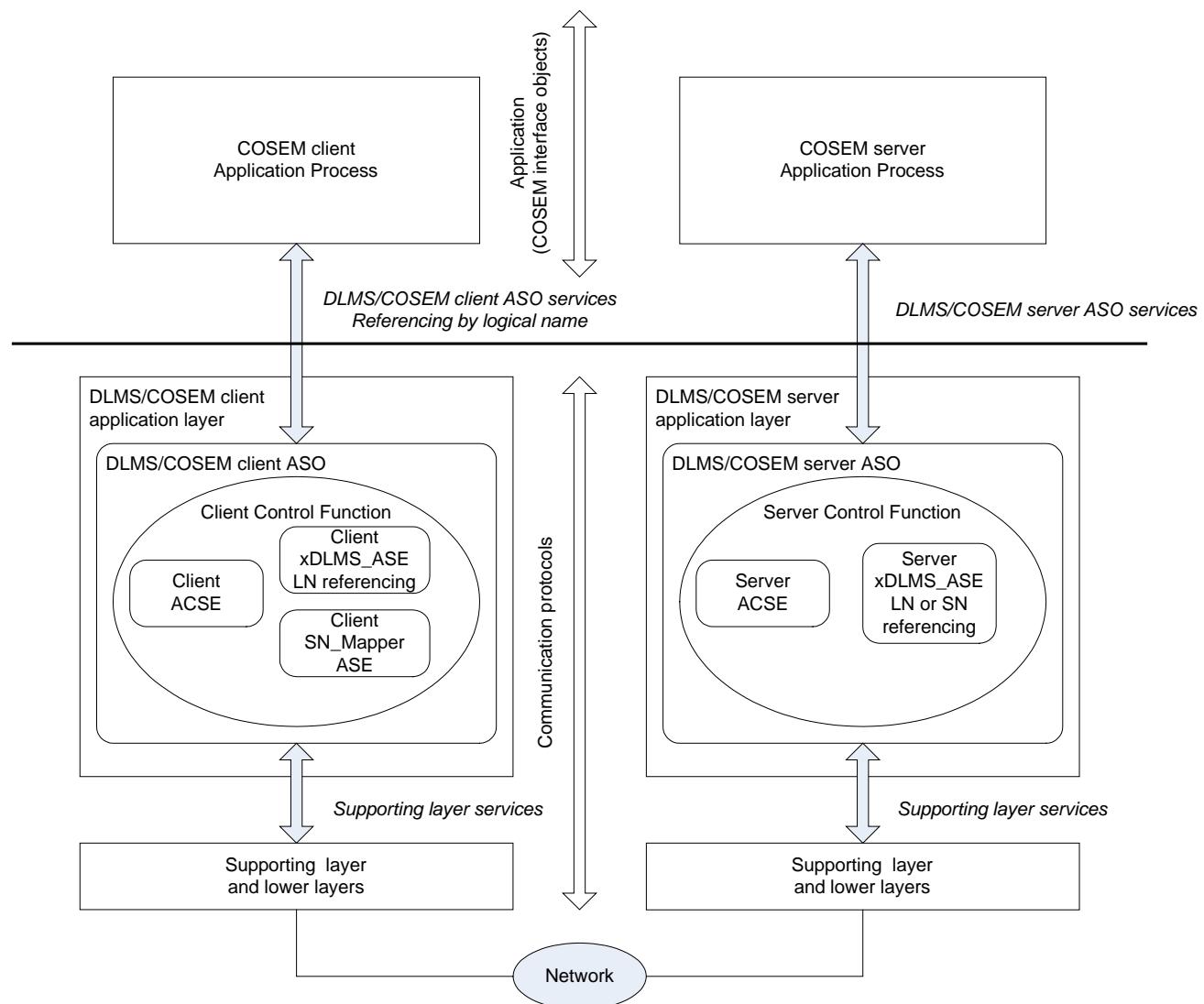


Figure 61 – The structure of the DLMS/COSEM application layers

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 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 layer. See also 9.4.1.

NOTE 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 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	2015-12-14	DLMS UA 1000-2 Ed. 8.1	153/500
-----------------------	------------	------------------------	---------

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 Ed. 12.1:2015, 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.

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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	154/500
-----------------------	------------	------------------------	---------

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, 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 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.

To support push operation, the DataNotification service is available, see 9.3.10. It can be used both 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 Ed. 12.1:2015, 4.4.8.

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 Ed. 12.1:2015 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 Ed. 12.1:2015, 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/COSEM 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.

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.

NOTE 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 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;
- service-specific block transfer mechanism. This mechanism is available with the GET, SET, ACTION, Read and Write services. In this case, the service primitive invocations contain only

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	157/500
-----------------------	------------	------------------------	---------

one part – one block – of the data (e.g. attribute values), so that the encoded form fits in a single APDU.

NOTE There is no block-recovery mechanism with 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 layer. Such APDUs may be transported if the supporting layer provide(s) segmentation; see Clause 10.

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 62. See also Figure 88.

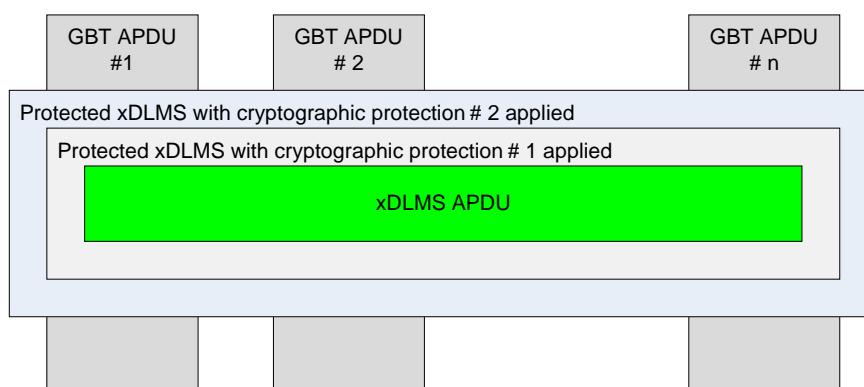


Figure 62 – The concept of composable xDLMS messages

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)

This mechanism can be used to transfer any xDLMS APDU in blocks. With GBT, the blocks are carried by general-block-transfer APDUs instead of service-specific "with-datablock" APDUs.

NOTE 1 The ACCESS and the DataNotification services do not provide a service-specific block transfer mechanism.

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 2 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 is not confirmed, it can be sent again. If streaming is used, lost block recovery takes place at the end of the streaming window.

The GBT mechanism is managed by the AL using the block transfer streaming parameters specified in 9.3.5.

Using the general block transfer mechanism is a negotiable feature, see 9.4.6.1.

The protocol of the general block transfer mechanism is specified in 9.4.6.13.

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.

9.1.4.7 xDLMS conformance block

The xDLMS conformance block enables optimised DLMS/COSEM 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 12 have been made. The xDLMS Initiate service uses these names for PDU sizes.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	159/500
-----------------------	------------	------------------------	---------

Table 12 – Clarification of the meaning of PDU size for DLMS/COSEM

was:	new:
Page 61, Table 3 of IEC 61334-4-41:1996:	
Proposed Max PDU Size	Client Max Receive PDU Size
Negotiated Max PDU Size	Server Max Receive PDU Size
Page 63, 5th paragraph of IEC 61334-4-41:1996	
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	<p>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.</p> <p>Values below 12 are reserved. The value 0 indicates that there is no limit on the PDU size.</p>
Page 63, last paragraph of IEC 61334-4-41:1996	
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 that the VDE-handler may support.	<p>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.</p> <p>Values below 12 are reserved. The value 0 indicates that there is no limit on the PDU size.</p>

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.

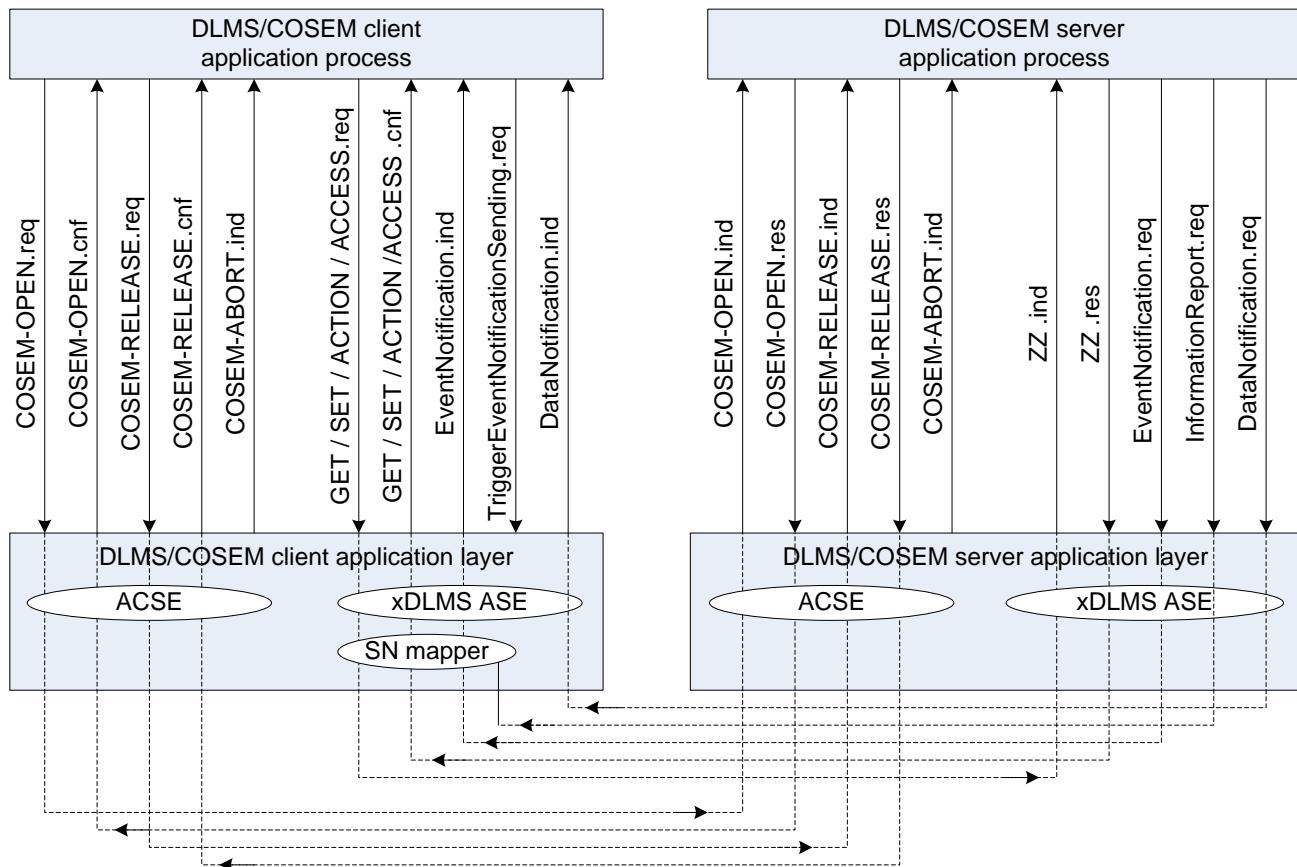
9.1.6 Summary of DLMS/COSEM application layer services

A summary of the services available at the top of the DLMS/COSEM AL is shown in Figure 63. 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 63 – 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/COSEM servers – COSEM object attributes and methods – can be accessed by DLMS/COSEM 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 and to log their activities accessing the meter. See also 4.3.6.

NOTE Client users may be operators or third parties.

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. There are three different authentication mechanisms available with different authentication security levels:

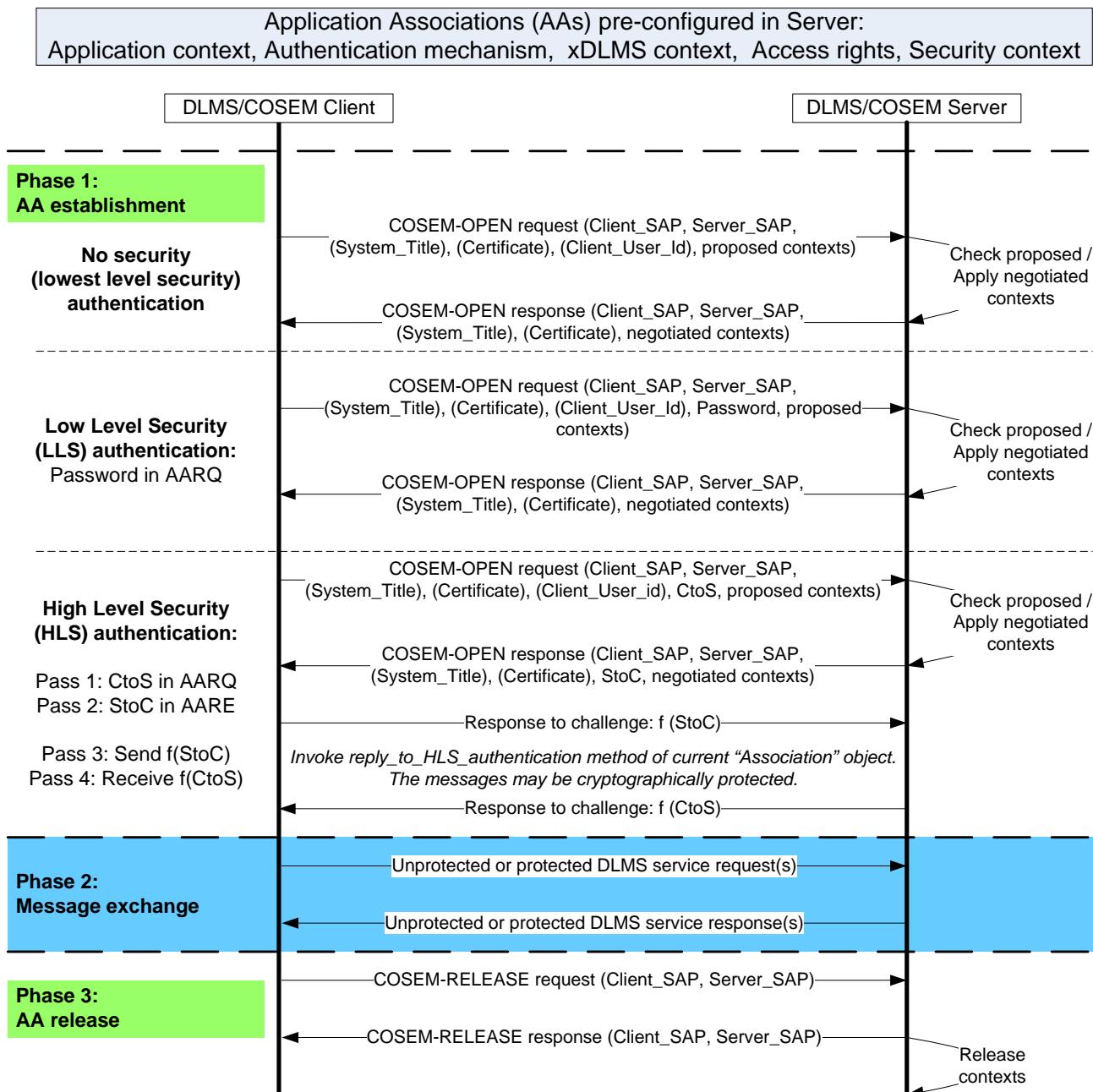
- no security (Lowest Level Security) authentication; see 9.2.2.2.2.2;
- Low Level Security (LLS) authentication, see 9.2.2.2.2.3;

NOTE 1 In ITU-T X.811 this is known as unilateral authentication, class 0 mechanism.

- High Level Security (HLS) authentication, see 9.2.2.2.2.4.

NOTE 2 In ITU-T X.811 this is known as mutual authentication using challenge mechanisms.

They are shown in Figure 64. Authentication mechanisms are identified by names, see 9.4.2.2.3.



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 64 – Authentication mechanisms

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.

9.2.2.2.2.2 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.

NOTE The dedicated-key, if transferred, can be used from this moment.

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*.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	164/500
-----------------------	------------	------------------------	---------

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 Ed. 12.1:2015 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 Ed. 12.1:2015 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.

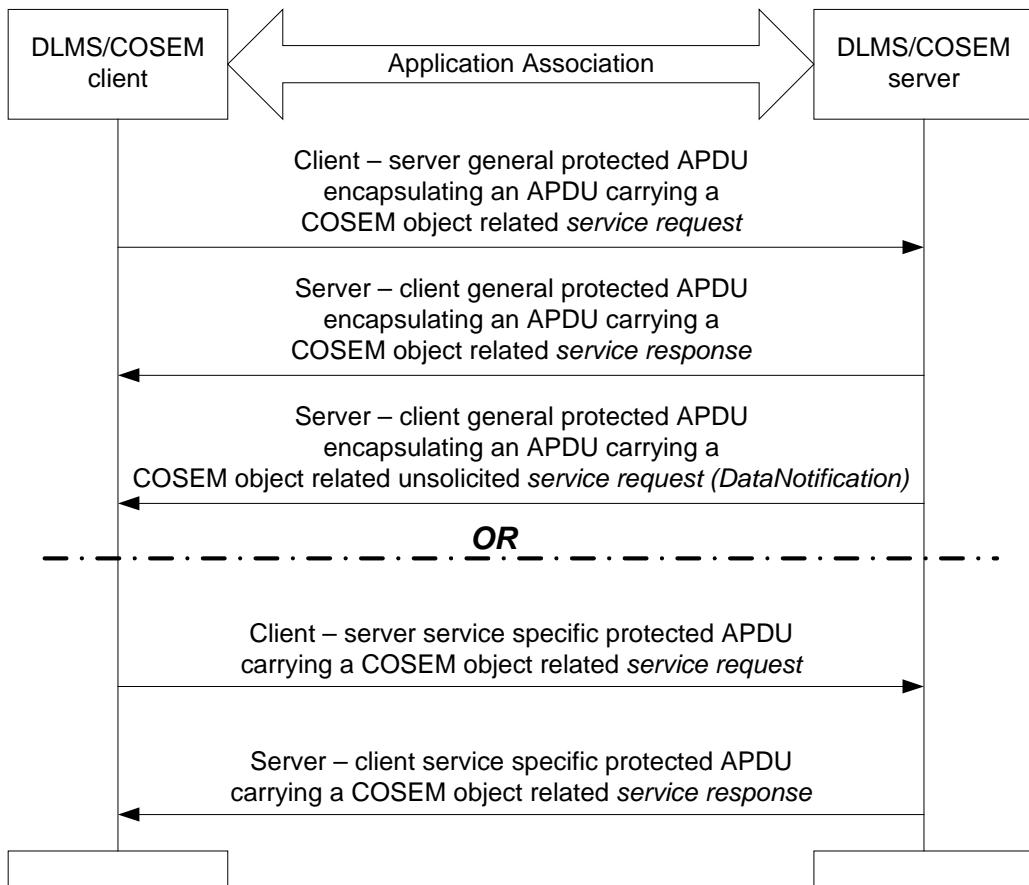
9.2.2.5 Application layer message security

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.

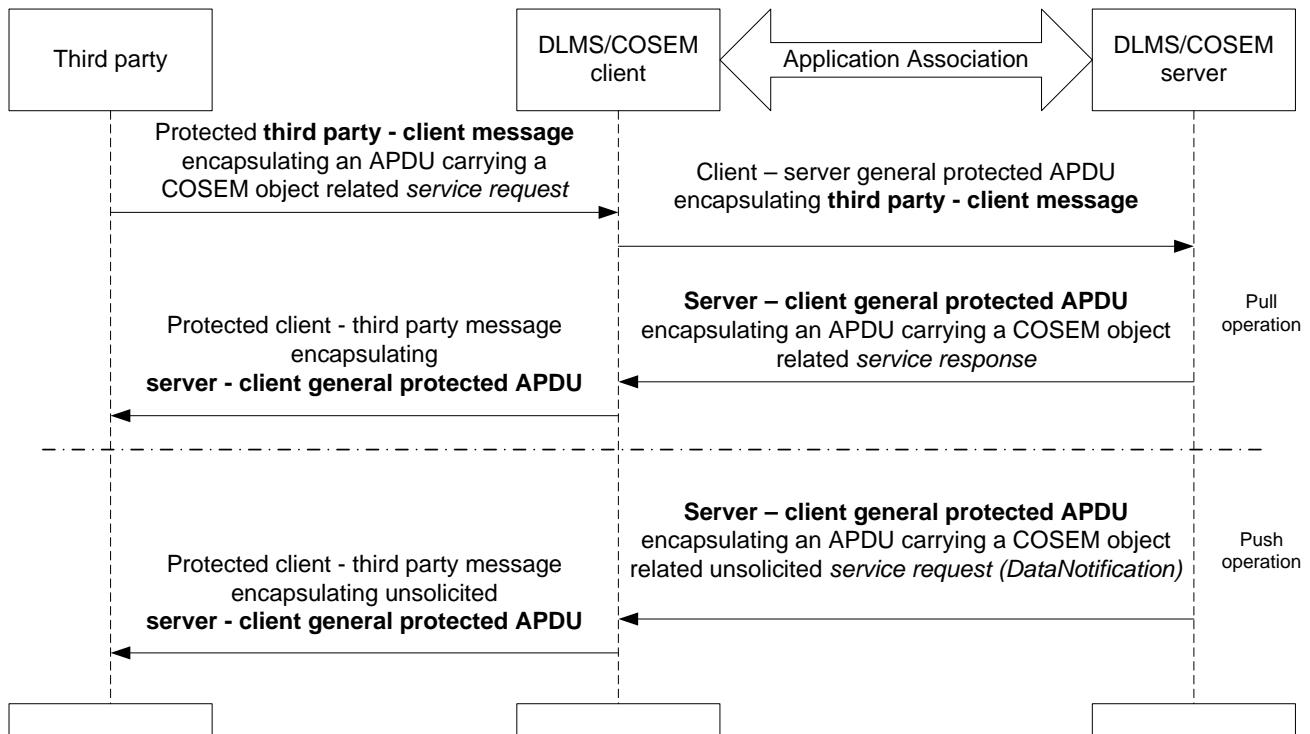
NOTE 1 Project specific companion specifications may specify additional criteria for accepting and processing messages.

The concept of message protection between a client and a server is shown in Figure 65.

**Figure 65 – 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/COSEM 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 66.

**Figure 66 – 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/COSEM 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.

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 Ed. 12.1:2015 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 800-21:2005:2005 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 800-21:2005:2005 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 67 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 (H_2) is equal to the retrieved or received hash value (H_1), then it can be assumed that the retrieved or received data (M_2) is the same as the original data (M_1) (i.e., $M_1 = M_2$).

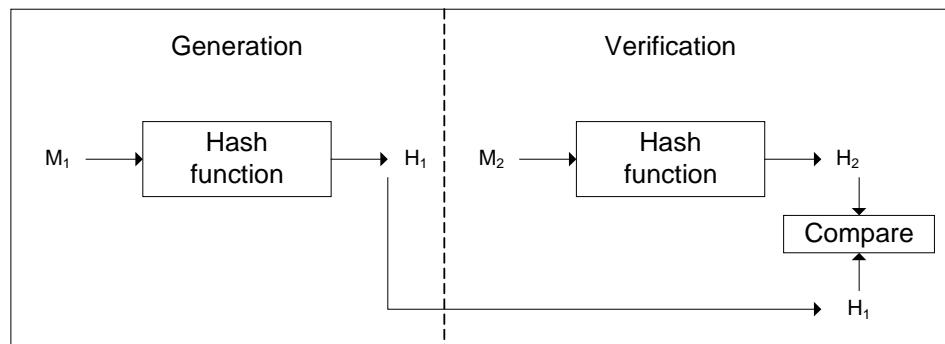


Figure 67 – Hash function

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 19.

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 800-21:2005:2005 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 800-21:2005:2005 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 68 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).

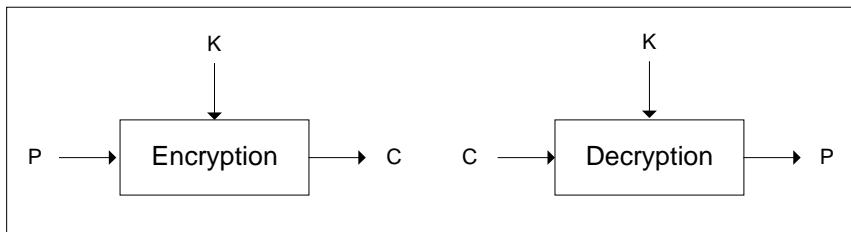


Figure 68 – Encryption and decryption

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 800-21:2005:2005 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 800-21:2005:2005 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 800-21:2005:2005 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 69 depicts the use of message authentication codes (MACs).

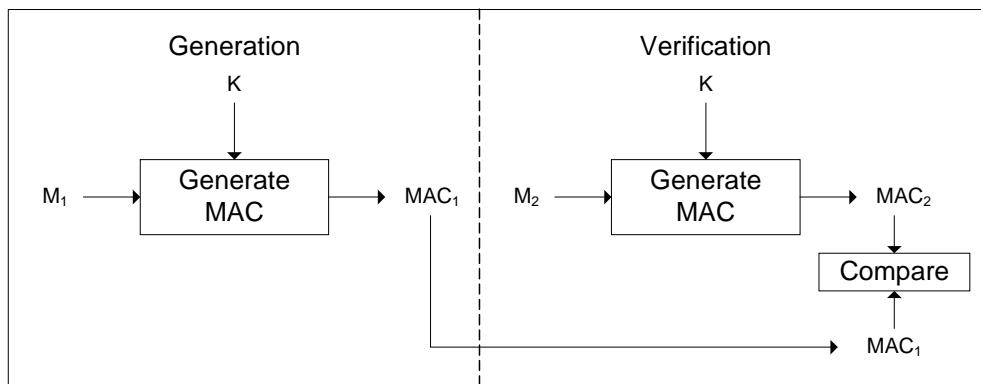


Figure 69 – Message Authentication Codes (MACs)

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 70.

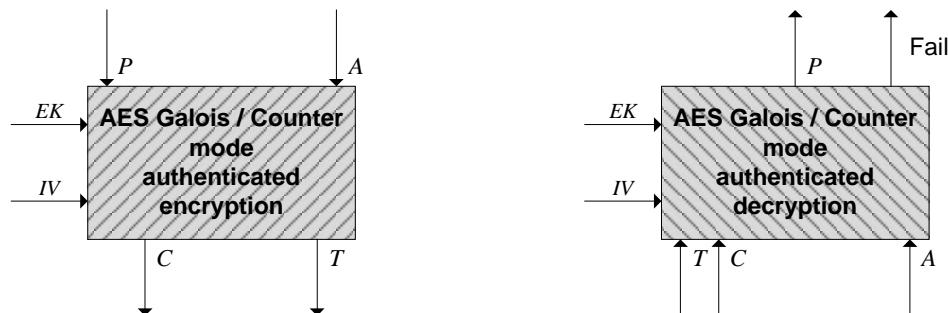


Figure 70 – GCM functions

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.

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 .

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.

For each encryption key EK (i.e. block cipher key) an invocation counter (IC) is maintained separately for the authenticated encryption and the authenticated decryption function. 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 then it is incremented by 1. However, when the maximum value of the IC has been reached, any following

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	173/500
-----------------------	------------	------------------------	---------

invocation 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. Verification of the *IC* fails – and with this, the authenticated decryption function fails – if the value being verified is smaller than the lowest acceptable value. If the verification is successful the lowest acceptable value is set to the value of the *IC* verified plus 1. If the value being verified is equal to the maximum value, the authenticated decryption function shall return an error.

NOTE The maximal 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): $\text{len}(KEK) = 256$.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	174/500
-----------------------	------------	------------------------	---------

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 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 13.

Table 13 – Elliptic curves in DLMS/COSEM security suites

DLMS security suite	Curve name in FIPS PUB 186-4	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:

- $x = x_{d-1} \cdot 256^{d-1} + x_{d-2} \cdot 256^{d-2} + \dots + x_1 \cdot 256 + x_0;$
- 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.

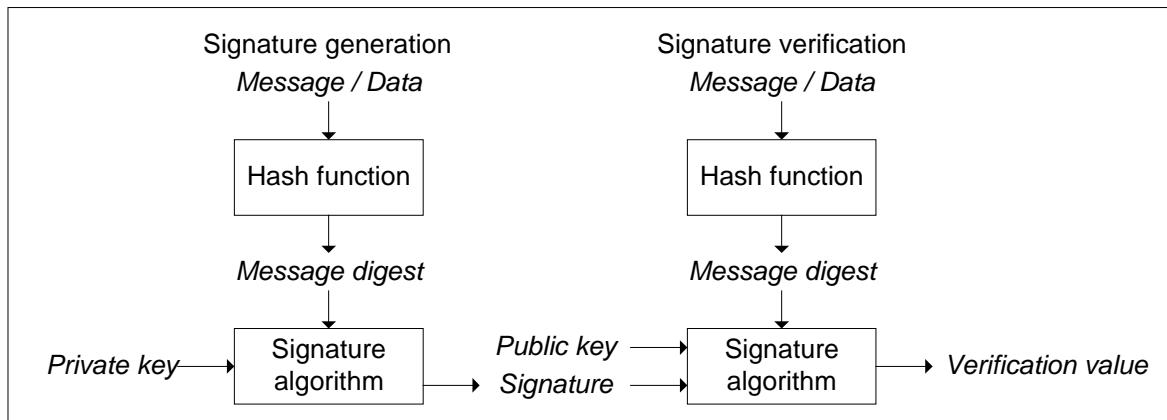
**Figure 71 – Digital signatures**

Figure 71 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 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.

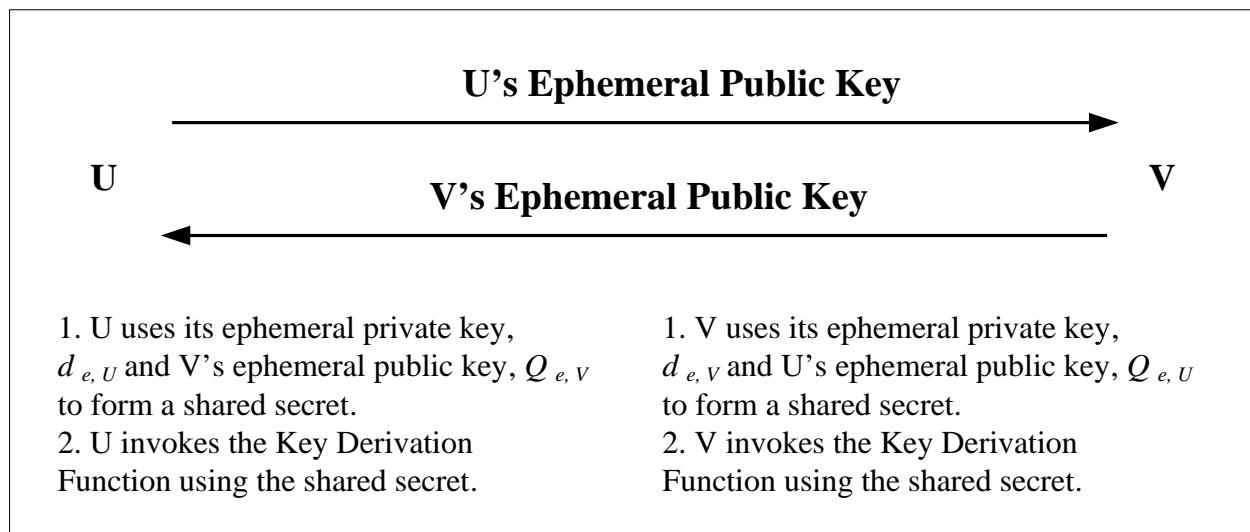
NOTE For NSA Suite B the first two schemes have been selected.

9.2.3.4.6.2 The Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme

This scheme is for use between a DLMS/COSEM 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 Ed. 12.1:2015 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 72 and Table 14 below.



NOTE This figure reproduces NSA2 Figure 1.

Figure 72 – C(2e, 0s) scheme: each party contributes only an ephemeral key pair

Prerequisites:

- 1) 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;
- 2) 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;
- 3) prior to or during the key agreement process, the parties obtain the identifier associated with the other party during the key agreement scheme.

NOTE 1 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 14 – Ephemeral Unified Model key agreement scheme summary

	Party U	Party V
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.

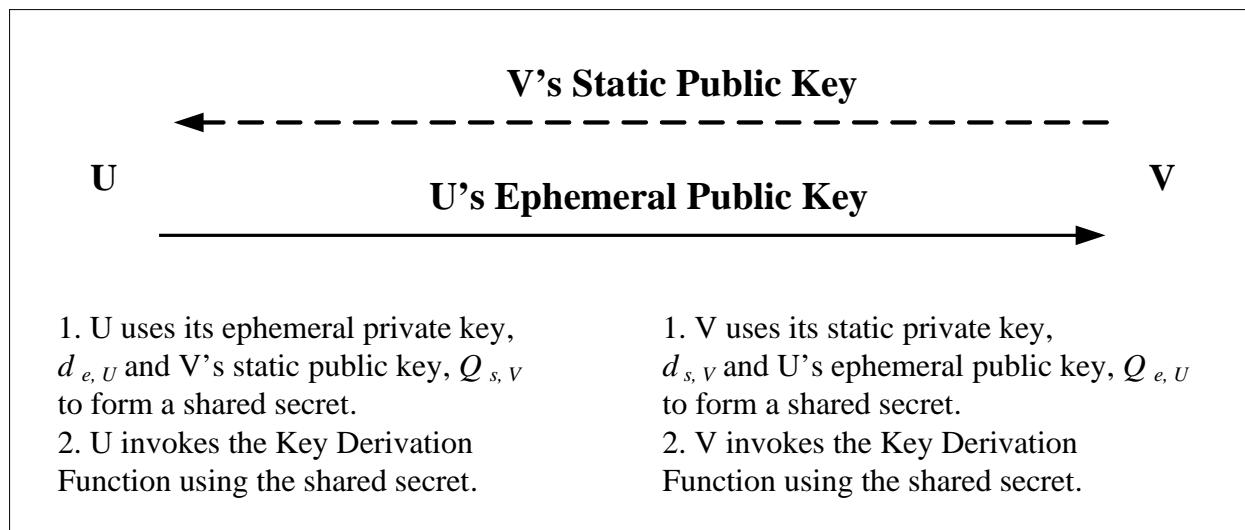
9.2.3.4.6.3 The One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) scheme

This scheme is for use by a DLMS/COSEM 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 73 and Table 15.



NOTE This figure reproduces NSA2 Figure 2.

Figure 73 – C(1e, 1s) schemes: party U contributes an ephemeral key pair, and party V contributes a static key pair

Prerequisites:

- 1) 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;
- 2) 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 ;
- 3) 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;
- 4) 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 15 – One-pass Diffie-Hellman key agreement scheme summary

	Party U	Party V
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 C.2.

9.2.3.4.6.4 The Static Unified Model C(0e, 2s, ECC CDH) scheme

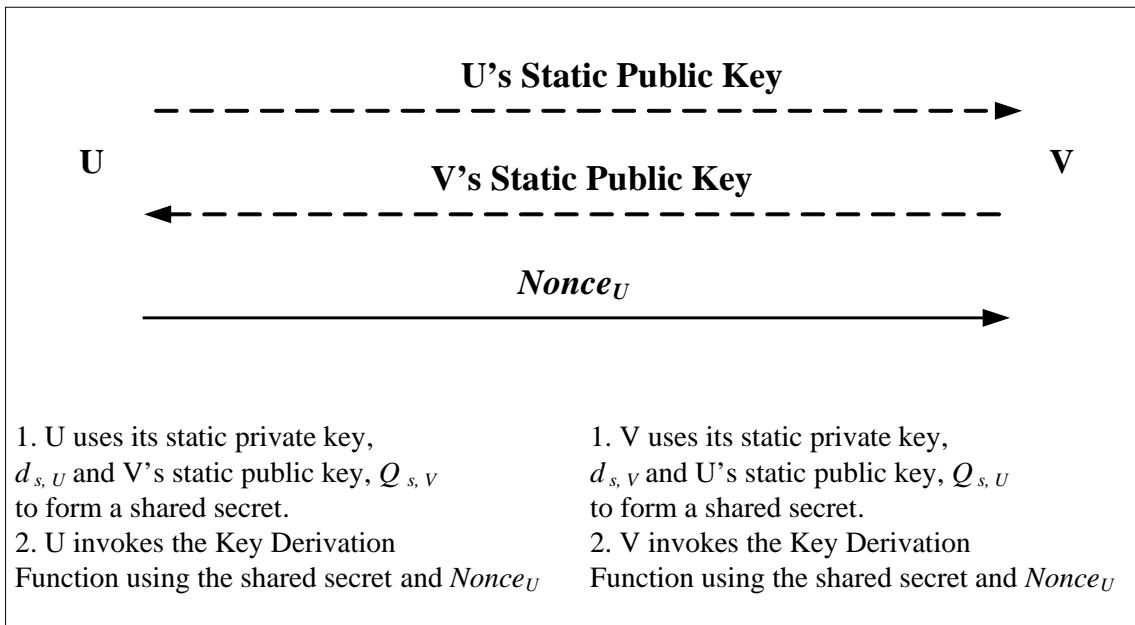
This scheme is for use by a DLMS/COSEM server and another party to agree on an ephemeral encryption key to protect xDLMS APDUs or COSEM data. The party sending the message (the originator) plays the role of party U and the other party (the recipient) plays the role of party V.

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 74** and **Table 16**.



NOTE This figure is based on NIST SP 800-56A Rev. 2: 2013 Figure 15.

Figure 74 – C(0e, 2s) scheme: each party contributes only a static key pair

Prerequisites:

- 1) 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;
- 2) 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 ;
- 3) 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;
- 4) prior to or during the key agreement process, each party shall obtain the identifier associated with the other party during the key agreement scheme;
- 5) 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 16 – Static Unified Model key agreement scheme summary

	Party U	Party V
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	$Nonce_U$	
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	1. Compute <i>DerivedKeyingMaterial</i> using $Nonce_U$ 2. Destroy Z	1. Compute <i>DerivedKeyingMaterial</i> using $Nonce_U$ 2. Destroy Z
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 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 NSA2:2009 Clause 5 shall be used. It is as follows:

Function call: kdf(Z, *OtherInput*)

where *OtherInput* consists of *keydatalen* and *OtherInfo*.

Implementation-Dependent Parameters:

- 1) *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;
- 2) *max_hash_inputlen*: an integer that indicates the maximum length (in bits) of the bit string(s) input to the hash function.

NOTE 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:

- 1) Z: a byte string that represents the shared secret z;
- 2) *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;
- 3) *OtherInfo*: A bit string equal to the following concatenation:

AlgorithmID || *PartyUInfo* || *PartyVInfo* {||*SuppPubInfo*} {||*SuppPrivInfo*}

where the subfields are defined as follows:

- 1) *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. See Table 18;
- 2) *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;
- 3) *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;
- 4) (Optional) *SuppPubInfo*: A bit string containing additional, mutually known public information. Not used in DLMS/COSEM;
- 5) (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 17.

Table 17 – 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 18
<i>PartyUInfo</i>	Fixed	Fixed	Variable	8+n	–
<i>ID_U</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_V</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 18. See also 9.4.2.2.4.

Table 18 – Security algorithm ID-s

Algorithm	COSEM cryptographic algorithm ID	Encoded value
AES-GCM-128	2.16.756.5.8.3.0	60 85 74 06 08 03 00
AES-GCM-256	2.16.756.5.8.3.1	60 85 74 06 08 03 01
AES-WRAP-128	2.16.756.5.8.3.2	60 85 74 06 08 03 02
AES-WRAP-256	2.16.756.5.8.3.3	60 85 74 06 08 03 03

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.

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. 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 19 – 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 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 19 – 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:

- their purpose:
 - 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;
 - an encryption key is used as the block cipher key of the AES-GCM algorithm, see also 9.2.3.3.7.4;
 - an authentication key is used as Additional Authenticated Data (AAD) in the AES-GCM algorithm, see also 9.2.3.3.7.5.
- their lifetime:
 - 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.
 - 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/COSEM client – server pair using one of the methods shown in Table 20. 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/COSEM 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 37 – is not relevant and shall be set to zero.

NOTE The AARQ and the AARE APDUs themselves are not protected.

Table 20 summarizes the symmetric key types, their purpose, the methods to establish them and their use with the different APDUs and between different entities.

Table 20 – Symmetric keys types

Key type	Purpose	Key establishment	Use
Master key, KEK ¹⁾	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 21
		Key wrapping	
		Key agreement ³⁾	
Global unicast encryption key, GUEK ²⁾	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 ³⁾	
Global broadcast encryption key, GBEK ²⁾	Block cipher key for broadcast - xDLMS APDUs and / or COSEM Data	Key wrapping	- "Data protection" object protection parameters
(Global) Authentication key, GAK ²⁾	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 ³⁾	
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 ⁴⁾	- 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 37; or
 – by the key-id parameter of the general-ciphering APDU, see Table 21;
 – or by the protection parameters of the "Data protection" IC, see DLMS UA 1000-1 Ed. 12.1:2015, 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 21 and further specified in 9.2.5.3, 9.2.5.4 and 9.2.5.5.

Table 21 – Key information with general-ciphering APDU and data protection

Key information choices		Comment
key-Info		
identified-key	S	The EK is identified
key-id	M	
global-unicast-encryption-key	S	GUEK
global-broadcast-encryption-key	S	GBEK
wrapped-key	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
agreed-key	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.
Legend: M: Mandatory (part of a SEQUENCE) S: Selectable (part of a CHOICE) For the ASN.1 specification, see 9.5.		
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.		

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 37 – 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 Ed. 12.1:2015, 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 Ed. 12.1:2015, 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 1 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;

NOTE 2 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 22 – are classified according to:

- their purpose: digital signature key or key agreement key;
- by their lifetime: static keys or ephemeral keys.

Table 22 – 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"> - on an xDLMS APDUs; and/or - on COSEM data; or - on an ephemeral public key agreement key. <p>Verifier uses public key to verify digital signature</p> <ul style="list-style-type: none"> - on an xDLMS APDUs; and/or - on COSEM data; or - on an ephemeral public key agreement key received.
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 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/COSEM 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 meter 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/COSEM 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/COSEM servers may also be provisioned with their own certificates and certificates of CAs, DLMS/COSEM 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 Ed. 12.1:2015, 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/COSEM 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/COSEM 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	2015-12-14	DLMS UA 1000-2 Ed. 8.1	191/500
-----------------------	------------	------------------------	---------

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.

For DLMS/COSEM a hierarchical PKI is foreseen, comprising the following components as shown in Figure 75.

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/COSEM clients, DLMS/COSEM servers and third parties: see 9.2.6.3.3.4.

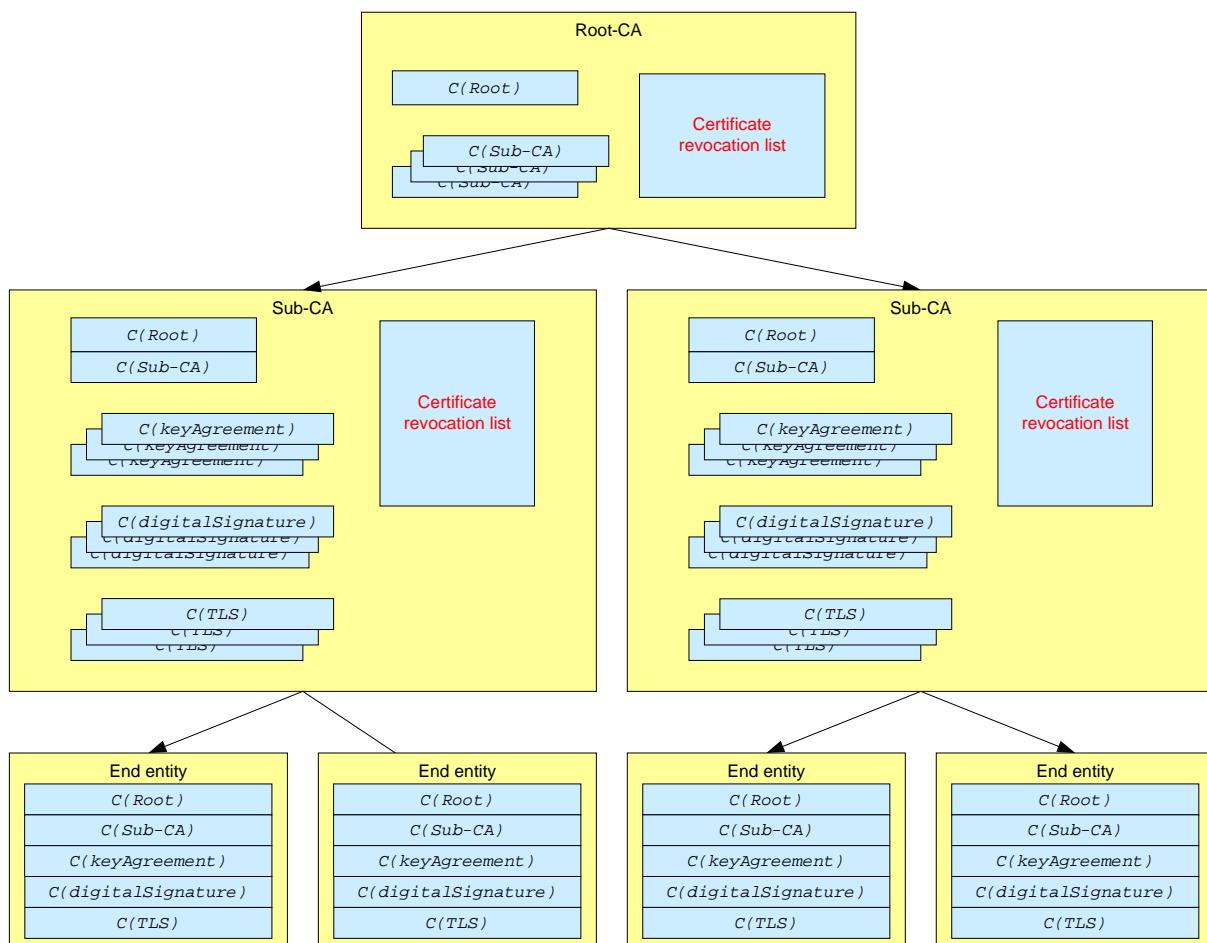


Figure 75 – Architecture of a Public Key Infrastructure (example)

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 meter market participants, meter 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.

9.2.6.3.3.4 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/COSEM client and a DLMS/COSEM 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 23.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	193/500
-----------------------	------------	------------------------	---------

Table 23 – 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 24. 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 24.

Table 24 – 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
Subject Unique ID	4.1.2.8	9.2.6.4.3.6	o	

Certificate field	RFC 5280 reference	Clause	m/x/o	Comment
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.

Table 25 – 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 26 – 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 27 – 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/COSEM servers `notBefore` and `notAfter` shall be well defined dates.

DLMS/COSEM 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 28.

Table 28 – X.509 v3 Certificate extensions

	Attributes	RFC 5280 reference	Clause	CAs		End entities		
				<i>C(Root)</i>	<i>C(Sub-CA)</i>	<i>C(TLS)</i>	<i>C(Key Agree)</i>	<i>C(Data Sign)</i>
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(Key Agree): Certificate an ECDH capable key establishment key C(Data Sign): Certificate of an ECDSA capable signing key								

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.

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).

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.

For the method of calculating the keyIdentifier see 9.2.6.4.4.2.

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 29.

Table 29 – Key Usage extensions

Certificate	$C(Root)$	$C(Sub-CA)$	$C(TLS)$	C (KeyAgree)	C (DataSign)
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-HardwareModuleName`) as defined in IETF RFC 4108. The `hwSerialNum` field shall be set to the system title.

- Value: See Table 30.

Table 30 – Subject Alternative Name values

Certificate	$C(Root)$	$C(Sub-CA)$	$C(TLS)$	$C(KeyAgree)$	$C(DataSign)$
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 31.

Table 31 – Issuer Alternative Name values

Certificate	$C(Root)$	$C(Sub-CA)$	$C(TLS)$	$C(KeyAgree)$	$C(DataSign)$
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 32.

Table 32 – Basic constraints extension values

Certificate	$C(Root)$	$C(Sub-CA)$	$C(TLS)$	$C(KeyAgree)$	$C(DataSign)$
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/COSEM 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/COSEM servers

Every DLMS/COSEM 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:

Table 33 – 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/COSEM 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/COSEM 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.

NOTE 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.

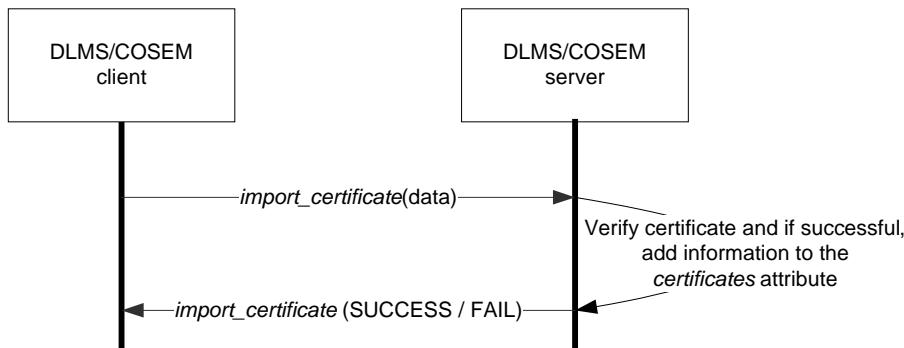
Directly trusted CA keys cannot be exported.

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 76.

Precondition: The DLMS/COSEM client verified the CA certificate.
The server has been provisioned with trust anchor(s).



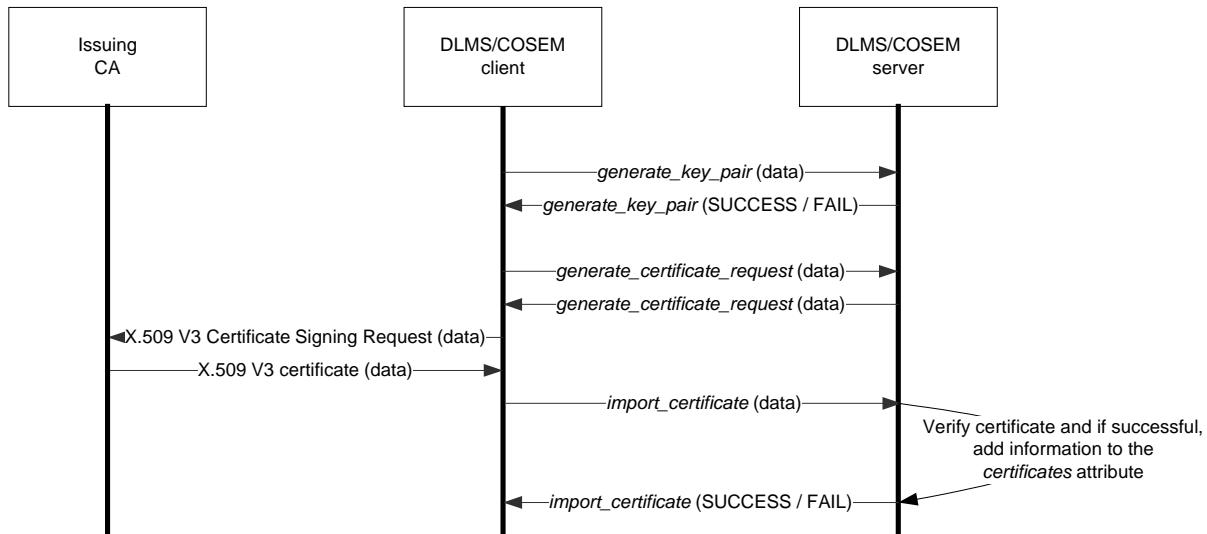
NOTE When a third party is responsible for managing CA certificates, then the *import_certificate* method may be invoked by that third party via the client acting as a broker.

Figure 76 – 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 77 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 77 – 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 X.509 v3 CSR, signed by the private key of the newly generated key pair;
- Step 3: The client sends the X.509 v3 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 Ed. 12.1:2015 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

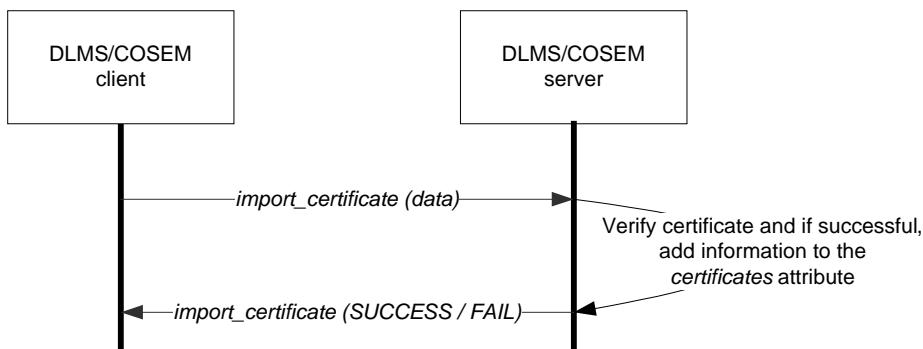
To verify digital signatures, to perform key agreement using a scheme that uses static key agreement keys, or to establish a TLS connection, the server needs to have the appropriate public key certificates of the other party.

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 Ed. 12.1:2015, 4.4.7. The process is shown in Figure 78.

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 78 – 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

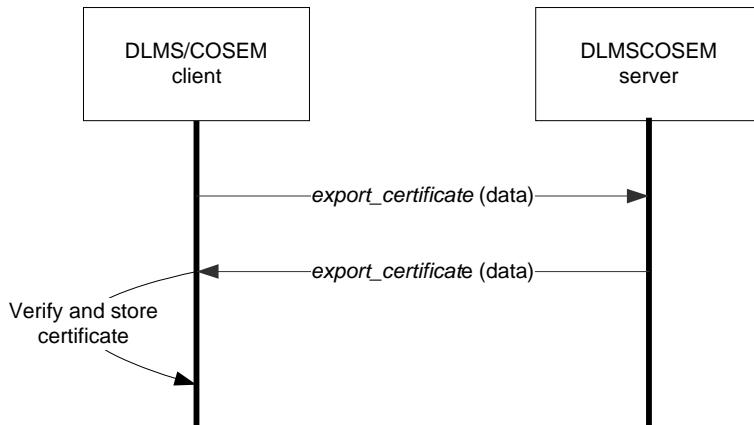
To verify digital signatures applied by the server, to perform key agreement that involves a static key agreement key, or to establish a TLS connection the client or third party needs to have the appropriate public key certificate of the server.

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 Ed. 12.1:2015 ,4.4.7. The process is shown in Figure 79.



NOTE The *export_certificate (data)* method may be also invoked by a third party, using the client as a broker.

Figure 79 – 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.

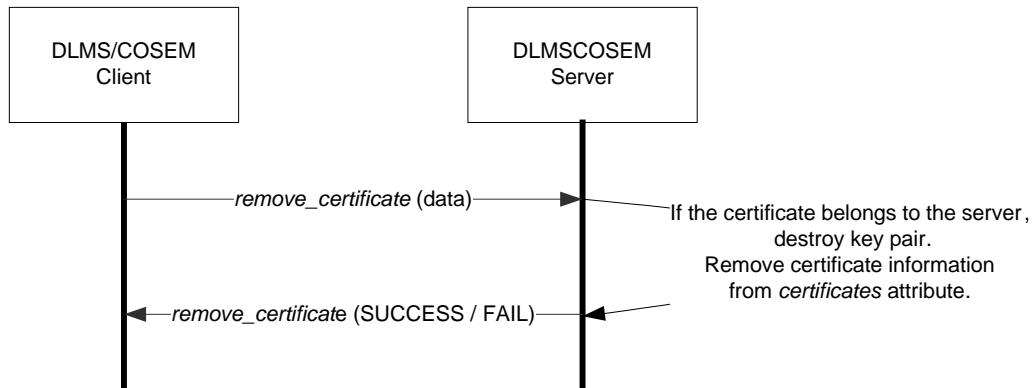
NOTE 2 The conditions when removal of a public key certificate is necessary are out of the Scope of this Technical Report.

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 80.



NOTE The *remove_certificate (data)* method may be also invoked by a third party, using the client as a broker.

Figure 80 – 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 Ed. 12.1:2015, 4.4.7 – the enum type shall be interpreted as an unsigned 8 type. The meaning of each bit is as shown in Table 34.

Table 34 – 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 Ed. 12.1:2015, 5.4.10. 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 Ed. 12.1:2015, 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 35.

For older versions, see their specification in DLMS UA 1000-1, the “Blue Book”.

Table 35 – Access rights values (“Association LN” ver 3 “Association SN” ver 4)

Bit	Attribute access	Method access
0	read-access	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 36. 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 36 – Ciphered xDLMS APDUs

APDU type	Parties	Type of ciphering	Security services	Key used	Com- pression
Service-specific glo-ciphering or ded-ciphering				Block cipher key: - Dedicated key ¹ - Global unicast / broadcast key established ² outside the exchange ³ , identified by the SC byte	–
general-glo-ciphering general-ded-ciphering	Client – Server	Symmetric key	Authentication Encryption	Authentication key: global, established ² outside the exchange ³	Yes ⁵
general-ciphering	Third party or Client – Server	Symmetric key	Authentication Encryption	Block cipher key: - Global unicast / broadcast, established ² outside the exchange ³ , identified as part of the exchange, or - Established ² as part of the exchange ⁴ Authentication key: global, established ² outside the exchange ³	Yes ⁵
general-signing		Asymmetric key	Digital signature	Signing key	No

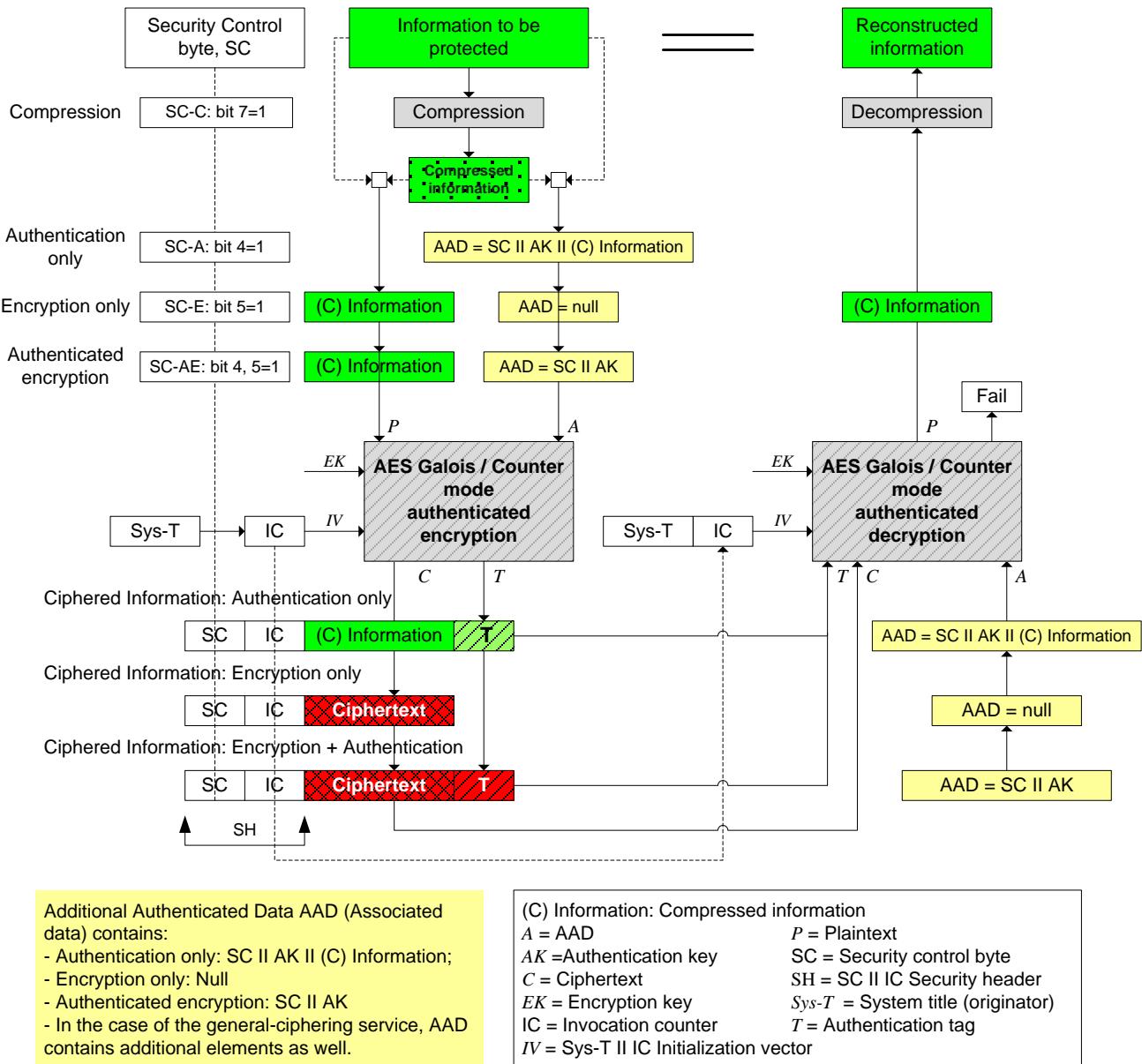
¹⁾ Transported by the AARQ;
²⁾ Key establishment may be key wrapping or key agreement;
³⁾ In the server, these keys are held by the Security setup objects;
⁴⁾ Key data is transported in the protected APDU;
⁵⁾ The use of compression is controlled by the Security Control byte.

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 81. 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).

**Figure 81 – 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 37 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 37 – 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 38, 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 38 – 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	-	$SC \parallel AK \parallel (C) I$	$SC \parallel AK \parallel$ transaction-id II ¹ originator-system-title II ¹ recipient-system-title II ¹ date-time II ¹ other-information II ¹ $(C) I$
1	0	Encrypted only	$(C) I$	-	-
1	1	Encrypted and authenticated	$(C) I$	$SC \parallel AK$	$SC \parallel AK \parallel$ transaction-id II ¹ originator-system-title II ¹ recipient-system-title II ¹ date-time II ¹ other-information II ¹

1) The fields transaction-idother-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

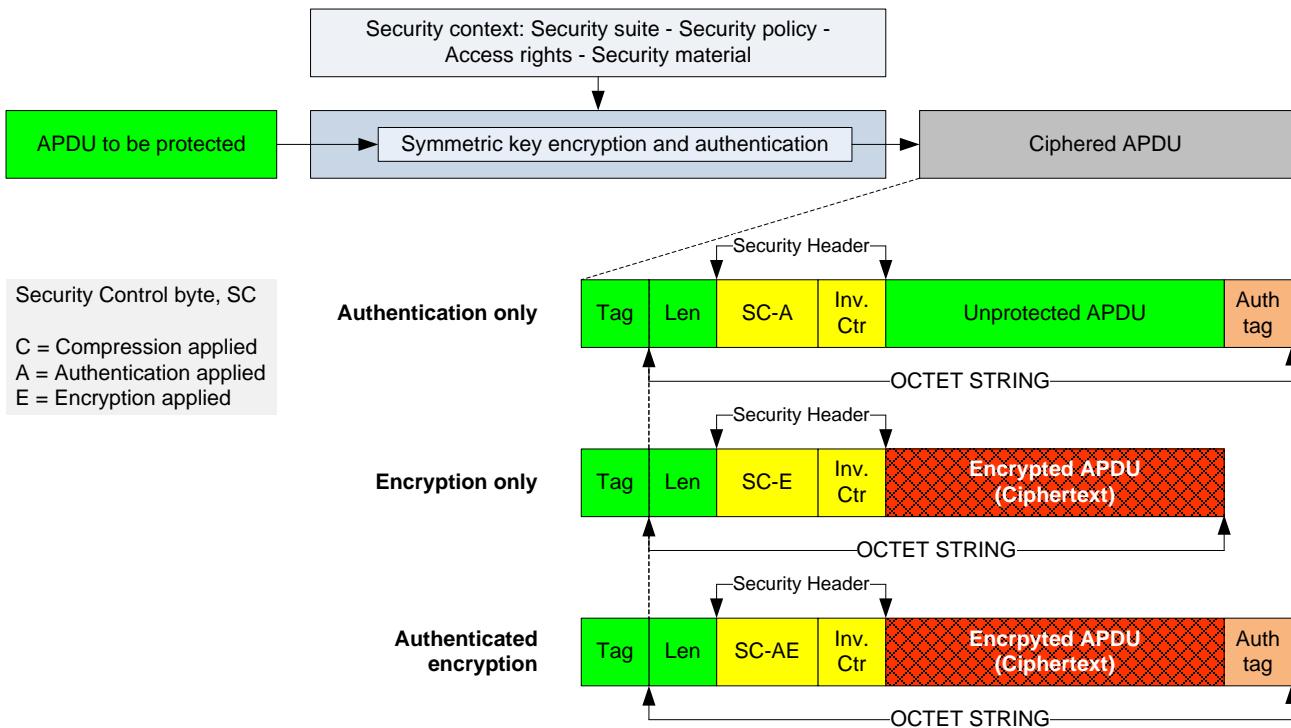
These keys used by AES-GCM are specified in 9.2.3.3.7.4 and 9.2.3.3.7.5 respectively. The various keys used in DLMS/COSEM and their establishment are specified in 9.2.5.

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 82. See also Table 38 and Table 39.

**Figure 82 – 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 83. See also Table 38 and Table 39.

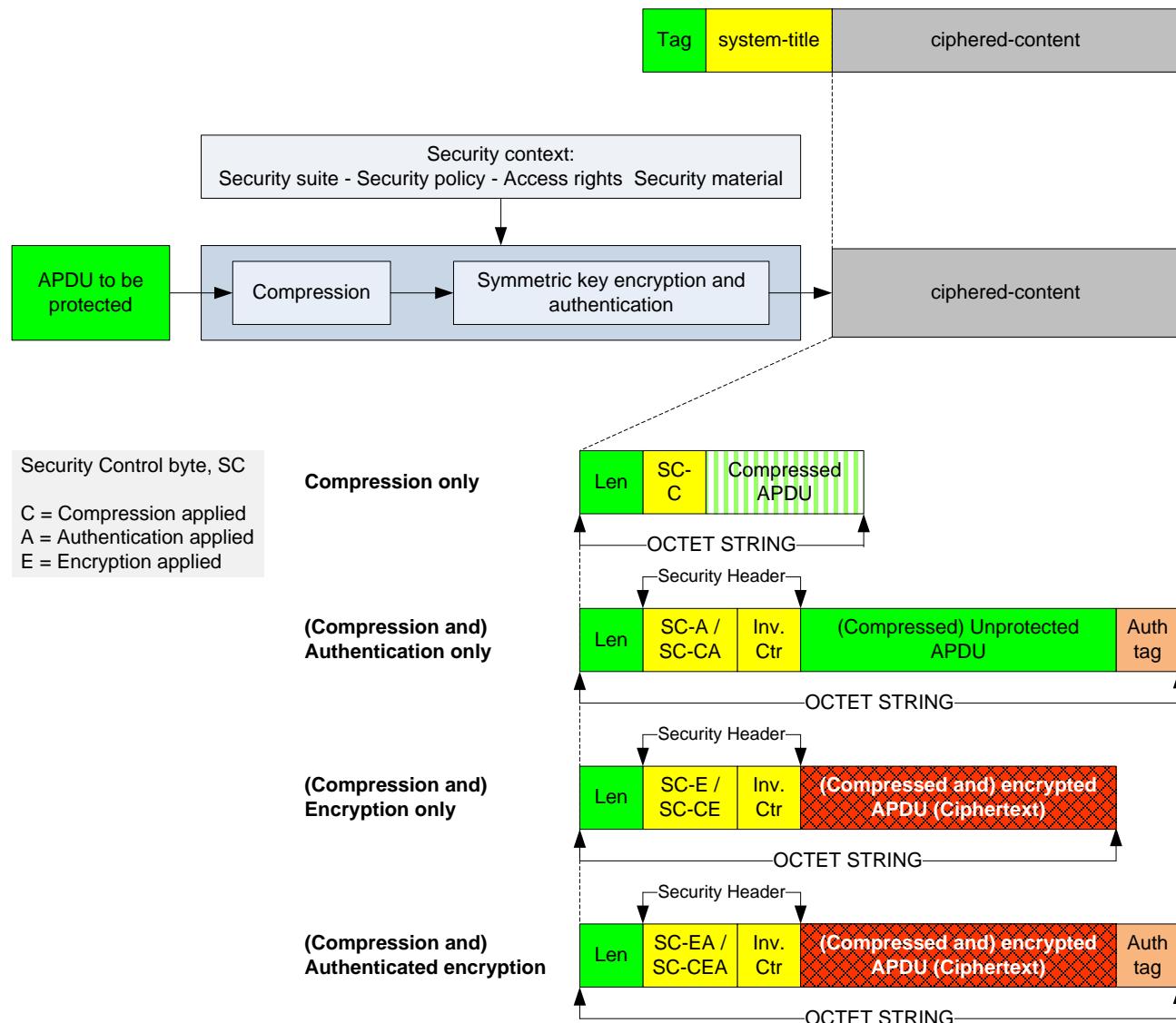


Figure 83 – Structure of general-glo-ciphering and general-ded-ciphering xDLMS APDUs

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 84. See also Table 38 and Table 39.

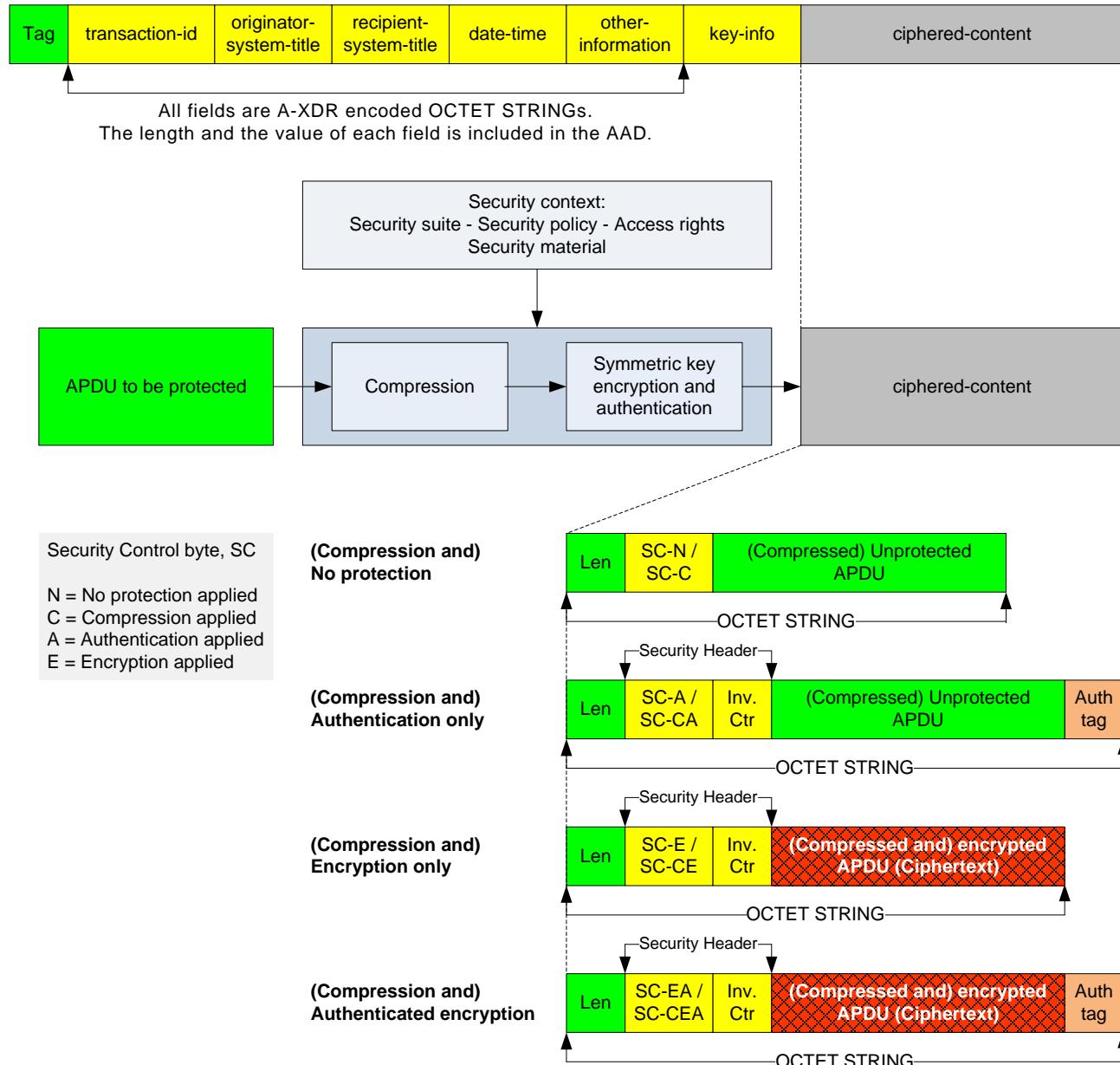


Figure 84 – Structure of general-ciphering xDLMS APDUs

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 39.

Table 39 – 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.
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 37. ¹⁾
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.
¹⁾ 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 40 shows an encoding example of a service-specific global ciphering xDLMS APDU: glo-get-request.

Table 40 – Example: glo-get-request xDLMS APDU

	X	Contents	LEN (X) bytes	Len (X) bits
Security material				
Security suite		GCM-AES-128		
System Title	Sys-T	4D4D4D0000BC614E (here, the five last octets contain the manufacturing number in hexa)	8	64
Invocation counter	IC	01234567	4	32
Initialization	IV	Sys-T II IC	12	96

	X	Contents			$LEN(X)$ bytes	$Len(X)$ bits
Vector		4D4D4D0000BC614E01234567				
Block cipher key (global)	EK	000102030405060708090A0B0C0D0E0F			16	128
Authentication Key	AK	D0D1D2D3D4D5D6D7D8D9DADBDCCDDDEDF			16	128
Security applied		Authentication	Encryption	Authenticated encryption		
Security control byte (with unicast key)	SC	$SC-A$ 10	$SC-E$ 20	$SC-AE$ 30	1	8
Security header	SH	$SH = SC-A \parallel IC$ 1001234567	$SH = SC-E \parallel IC$ 2001234567	$SH = SC-AE \parallel IC$ 3001234567	5	40
Inputs		Authentication	Encryption	Authenticated encryption		
xDLMS APDU to be protected	$APDU$	C0010000080000010000FF0200 (Get-request, attribute 2 of the Clock object)			13	104
Plaintext	P	Null	C001000008000001 0000FF0200	C001000008000001 0000FF0200	13	104
Associated data	A	$SC \parallel AK \parallel APDU$	–	$SC \parallel AK$		
Associated Data – Authentication	$A-A$	10D0D1D2D3D4D5D6 D7D8D9DADBDCCDDDE DFC001000080000 010000FF0200	–	–	30	240
Associated Data – Encryption	$A-E$	–	–	–	0	0
Associated Data – Authenticated encryption	$A-AE$	–	–	30D0D1D2D3D4D5D6 D7D8D9DADBDCCDDDE DF	17	136
Outputs		Authentication	Encryption	Authenticated encryption		
Ciphertext	C	NULL	411312FF935A4756 6827C467BC	411312FF935A4756 6827C467BC	13	104
Authentication tag	T	06725D910F9221D2 63877516	–	7D825C3BE4A77C3F CC056B6B	12	96
The complete Ciphered APDU		$TAG \parallel LEN \parallel SH \parallel APDU \parallel T$	$TAG \parallel LEN \parallel SH \parallel C$	$TAG \parallel LEN \parallel SH \parallel C \parallel T$	–	–
Authenticated APDU		C81E1001234567C0 010000080000100 00FF02006725D91 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 41 shows an example where the ACCESS.request and ACCESS.response APDUs shown in Table 133 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 40.

Table 41 – ACCESS service with general-ciphering, One-Pass Diffie-Hellman C(1e, 1s, ECC CDH) key agreement scheme

Message Elements	Contents	LEN (Bytes)
General-Ciphering	DD	1
transaction-id		0
length	08	1
value	0102030405060708	8
originator-system-title		0
length	08	1
value	4D4D4D0000BC614E	8
recipient-system-title		0
length	08	1
value	4D4D4D0000000001	8
date-time		0
length	00	1
value		0
other-information		0
length	00	1
value		0
key-info OPTIONAL	01	1
agreed-key CHOICE	02	1
key-parameters		0
length	01	1
value	01	1
key-ciphered-data		0
length	8180	2
value	C323C2BD45711DE4688637D919F92E9D B8FB2DFC213A88D21C9DC8DCBA917D81 70511DE1BADB360D50058F794B0960AE 11FA28D392CFF907A62D13E3357B1DC0 B51BE089D0B682863B2217201E73A1A9 031968A9B4121DCBC3281A69739AF874 29F5B3AC5471E7B6A04A2C0F2F8A25FD 772A317DF97FC5463FEAC248EB8AB8BE	128
ciphered-content		0
length	81EB	2
value	30000000003F14FC102AE08241BC24EF 12AA8049F2C6C81B9248C47AD8DC4BB6 BBE2DF0C96FDFA3722909D156F2A0F02 128C7CC404A0CE05898D077712D8997F AA390405C711FA771740D16290D73B11 A2431843D9D4AE3A3CD6B0EF5A811F9F 979A90AE1937BFF27084C4169EA91BE3 29D4893895AF46668E8C3938D40041F9 A7859EB6F78FABF95FDD4465C5E0975B 818FE1AA859514D299C1E0CDB7F3953F C5EE5598CEEBCDA138CC9BD9E108CEEEF 403ADC313C05510169F623147F4DDB2C 5120B9FB77BD511DD749A62CABAB3A4C A05F8136D33DC492C7C899649E07DF3F C30D61E81FBF8AC86EC3F9	235
ACCESS.request with authenticated encryption SC II IC II ciphertext II auth. Tag		

general-ciphering(encoded)	DD080102030405060708084D4D4D0000 BC614E084D4D4D00000000100000102 01018180C323C2BD45711DE4688637D9 19F92E9DB8FB2DFC213A88D21C9DC8DC BA917D8170511DE1BADB360D50058F79 4B0960AE11FA28D392CFF907A62D13E3 357B1DC0B51BE089D0B682863B221720 1E73A1A9031968A9B4121DCBC3281A69 739AF87429F5B3AC5471E7B6A04A2C0F 2F8A25FD772A317DF97FC5463FEAC248 EB8AB8E81EB30000000003F14FC102A E08241BC24EF12AA8049F2C6C81B9248 C47AD8DC4BB6BBE2DF0C96FDFA372290 9D156F2A0F02128C7CC404A0CE05898D 077712D8997FAA390405C711FA771740 D16290D73B11A2431843D9D4AE3A3CD6 B0EF5A811F9F979A90AE1937BFF27084 C4169EA91BE329D4893895AF46668E8C 3938D40041F9A7859EB6F78FABF95FDD 4465C5E0975B818FE1AA859514D299C1 E0CDB7F3953FC5EE5598CEEBDA138CC9 BD9E108CEEFF403ADC313C05510169F6 23147F4DDB2C5120B9FB77BD511DD749 A62CABAB3A4CA05F8136D33DC492C7C8 99649E07DF3FC30D61E81FBF8AC86EC3 F9	401
General-Ciphering	DD	1
transaction-id		0
length	08	1
value	0123456789012345	8
originator-system-title		0
length	08	1
value	4D4D4D0000000001	8
recipient-system-title		0
length	08	1
value	4D4D4D0000BC614E	8
date-time OPTIONAL		0
length	00	1
value		0
other-information		0
length	00	1
value		0
key-info OPTIONAL	01	1
agreed-key CHOICE	02	1
key-parameters		0
length	01	1
value	01	1
key-ciphered-data		0
length	8180	2
value	6439724714B47CD9CB988897D8424AB9 46DCD083D37A954637616011B9C23787 73295F0F850D8DAFD1B8E9FE666E53E4 F097CD10B38B69622152724A90987444 E1FF47974A1F6931A6502F58147463F0 E8CC517D47F55B0AC56DD8AC5C9D0E48 1934F2D90F9893016BD82B6E3FFE21FF 1588F3278B4E9D98EB4FB62ADD64B380	128

ciphered-content		0
length	3D	1
value ACCESS.response with authenticated encryption SC II IC II ciphertext II auth. tag	30000000000D168C67323F1484B77BA50F35F8693A4D9BFF236B71F10ABE857B6D22E1E235E30D50521552BBD97D0D99A766D151C7AA5164D78918F20C	61
general-ciphering (encoded)	DD080123456789012345084D4D4D000000000102010181806439724714B47CD9CB988897D8424AB946DCD083D37A954637616011B9C2378773295F0F850D8DAFD1BBE9FE666E53E4F097CD10B38B69622152724A90987444E1F47974A1F6931A6502F58147463F0E8CC517D47F55B0AC56DD8AC5C9D0E481934F2D90F9893016BD82B6E3FFE21FF1588F3278B4E9D98EB4FB62ADD64B3803D300000000000D168C67323F1484B77BA50F35F8693A4D9BFF236B71F10ABE857B6D22E1E235E30D50521552BBD97D0D99A766D151C7AA5164D78918F20C	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 85. For the additional fields, see Table 39 and 9.3.5.

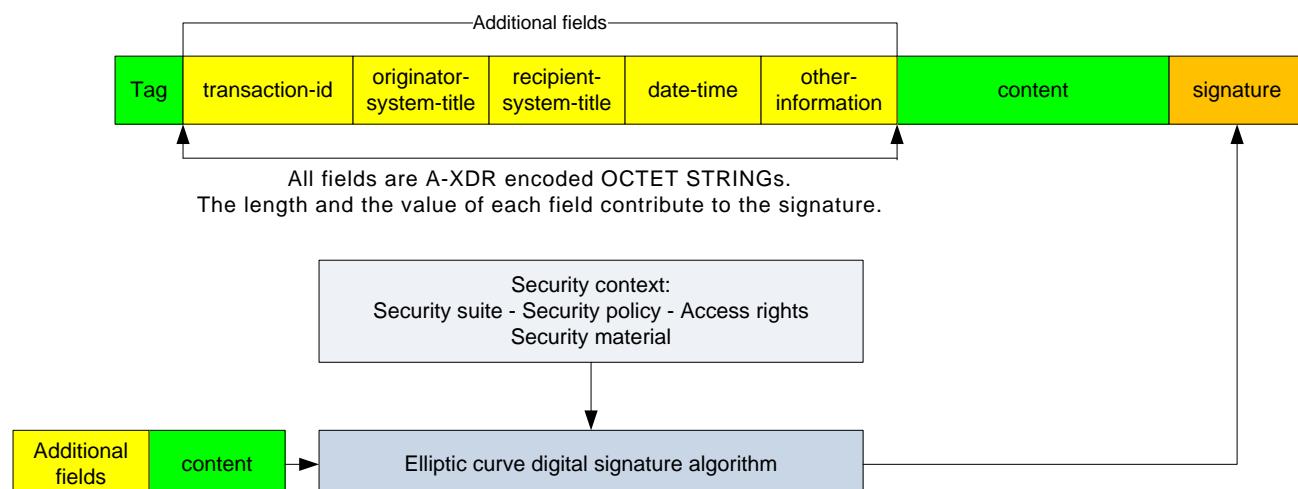


Figure 85 – Structure of general-signing APDUs

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 include the Originator_System_Title and the Recipient_System_Title, identifying the party that applied the protection 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 42.

Table 42 – 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 ¹	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 ¹			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 II 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 II CtoS)	ECDSA(SystemTitle-S SystemTitle-C CtoS II StoC)

Legend:
- C: Client, S: Server, CtoS: Challenge client to server, StoC: Challenge server to client
- IC: Invocation counter
- xx.request / .response: xDLMS service primitives used to access the <i>reply_to_HLS authentication</i> method of the "Association SN / LN" object.
¹ 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 64. 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 43 provides a test vector for HLS authentication-mechanism 5 with GMAC.

Table 43 – 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	Client	Server		
		4D4D4D0000000001	4D4D4D0000BC614E		
		(here, the five last octets contain the manufacturing number in hexa)		8	64
Invocation counter	IC	00000001	01234567	4	32
Initialization Vector	IV	Sys-T II IC		12	96
		Client	Server		
		4D4D4D0000000001 00000001	4D4D4D0000BC614E 01234567		
Block cipher key (global)	EK	000102030405060708090A0B0C0D0E0F		16	128
Authentication Key	AK	D0D1D2D3D4D5D6D7D8D9DADBDCCDDDEF		16	128
Security control byte	SC	10		1	8
Pass 1: Client sends challenge to server					
CtoS		4B35366956616759 "K56iVagY"		8	64
Pass 2: Server sends challenge to client					
StoC		503677524A323146 "P6wRJ21F"		8	64
Pass 3: Client processes StoC					
SC II AK II StoC		10D0D1D2D3D4D5D6D7D8D9DADBDCCDDDEF5036775 24A323146			
T = GMAC (SC II AK II StoC)		1A52FE7DD3E72748973C1E28		12	96
f(StoC) = SC II IC II T		10000000011A52FE7DD3E72748973C1E28		17	136
Pass 4: Server processes CtoS					
SC II AK II CtoS		10D0D1D2D3D4D5D6D7D8D9DADBDCCDDDEF4B35366 956616759			
T (SC II AK II CtoS)		FE1466AFB3DBCD4F9389E2B7		12	96
f(CtoS) = SC II IC II T		1001234567FE1466AFB3DBCD4F9389E2B7		17	136

Table 44 provides a test vector for HLS authentication-mechanism 7 with ECDSA.

Table 44 – 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)	C5C6D6620DBE1A39FCE50F4D64F0DB712D6FB57 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) 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 Ed. 12.1:2015, 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 86 illustrates this notion of service hierarchy and shows the relationship of the two correspondent N-users and their associated N-layer peer protocol entities.

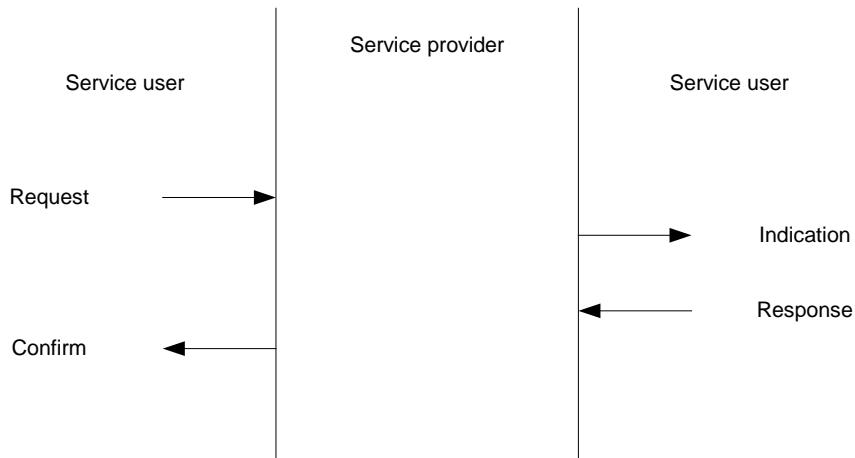


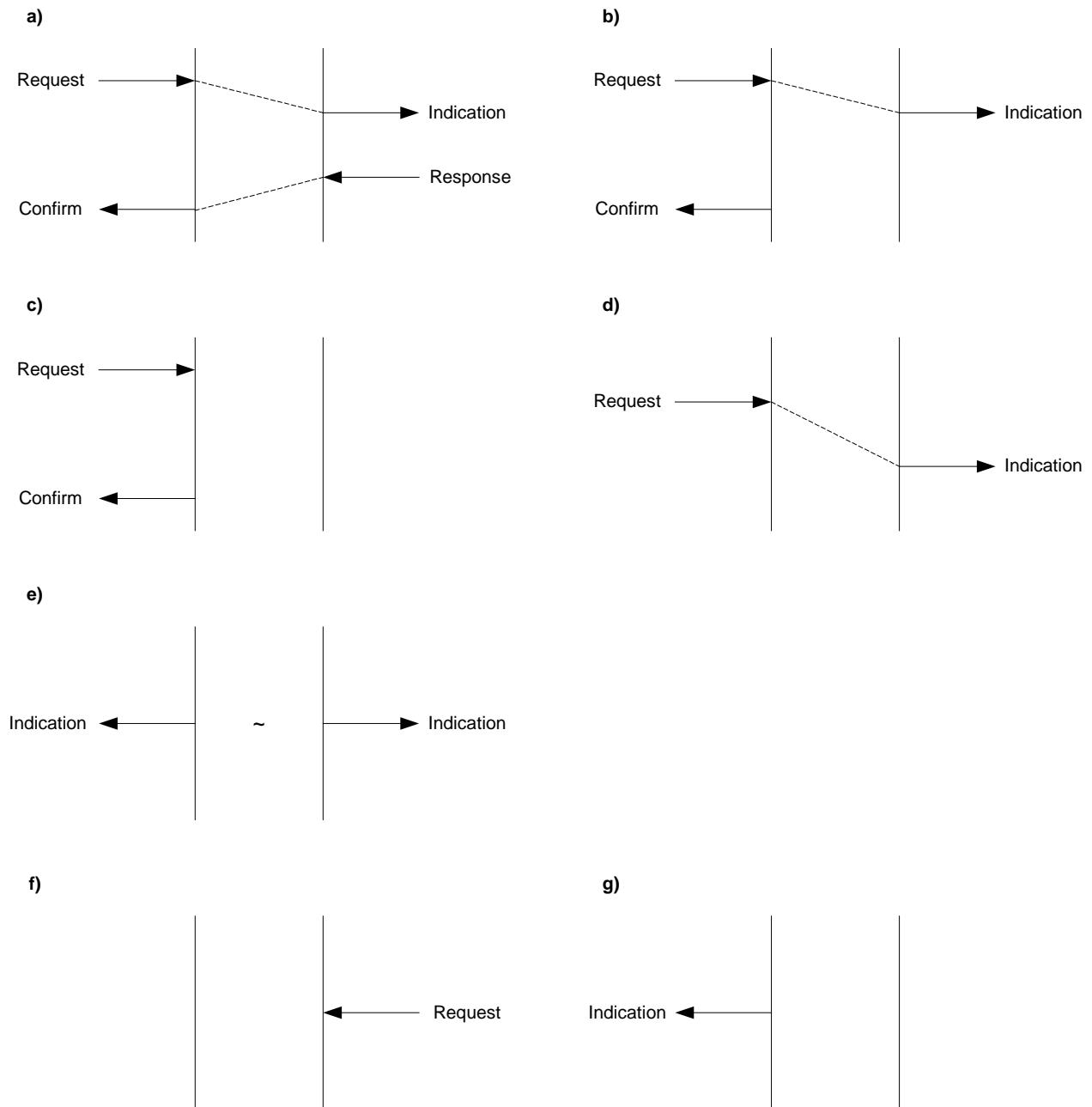
Figure 86 – Service primitives

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 87. 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 87 – 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 45.

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 45 – 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

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.

Semantics of the service primitives

The COSEM-OPEN service primitives shall provide parameters as shown in Table 46.

Table 46 – 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 Application_Context_Name indicates an application context using ciphering, it may carry the client system title specified in 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 Ed. 12.1:2015, 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 Application_Context_Name parameter indicates an application context using ciphering, it may carry the server system title specified in 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	225/500
-----------------------	------------	------------------------	---------

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 be present only, 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 present, the xDLMS InitiateRequest APDU shall be 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 12.

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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	226/500
-----------------------	------------	------------------------	---------

The Server_Max_Receive_PDU_Size element carries the maximum length of the xDLMS APDUs the server can receive. See Table 12.

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 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.

Use

Possible logical sequences of the COSEM-OPEN service primitives are illustrated in Figure 87.

- 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	227/500
-----------------------	------------	------------------------	---------

9.3.3 The COSEM-RELEASE service

Function

The function of the COSEM-RELEASE service is to gracefully release an existing AA. Depending on the way it is invoked, it uses the A-RELEASE service of the ACSE or not.

Semantics of the service primitives

The COSEM-RELEASE service primitives shall provide parameters as shown in Table 47.

Table 47 – 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 (=)

The Use_RLRQ_RLRE parameter in the .request primitive is optional. If present, its value may be FALSE (default) or TRUE. It indicates whether the ACSE A-RELEASE service – involving an RLRQ / RLRE APDU exchange – should be used or not. The A-RELEASE service and the RLRQ / RLRE APDUs are specified in 9.4.2. 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. Otherwise, it shall not be present 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 layer of the AL.

The Reason parameter 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 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.

The Proposed_xDLMS_Context parameter 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 securing the COSEM-RELEASE service, and avoiding thereby a denial-of-service attack that may be carried out by unauthorized releasing of the AA.

In the .request primitive, the Proposed_xDLMS_Context parameter 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.

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.

Otherwise, the RLRQ APDU is silently discarded.

The Local_or_Remote parameter 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.

The Result parameter is mandatory. In the .response primitive, it indicates 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.

The Failure_Type parameter 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.

The User_Information parameter in the .request primitive is optional. If present, it is passed to the supporting 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). Similarly, it is optional in the .response primitive. If present, it is passed to the supporting 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).

The specification of the content of the User_Information parameter is not within the Scope of this Technical Report. See also Clause 10.

Use

Possible logical sequences of the COSEM-RELEASE service primitives are illustrated in Figure 87:

- 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 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	229/500
-----------------------	------------	------------------------	---------

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 layer is disconnected; or
- remotely, when a RLRE APDU is received. The supporting 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.

9.3.4 The COSEM-ABORT service

Function

The function of the COSEM-ABORT service is to indicate an unsolicited disconnection of the supporting layer.

Semantics

The COSEM-ABORT service primitives shall provide parameters as shown in Table 48.

Table 48 – Service parameters of the COSEM-ABORT service primitives

	.indication
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.

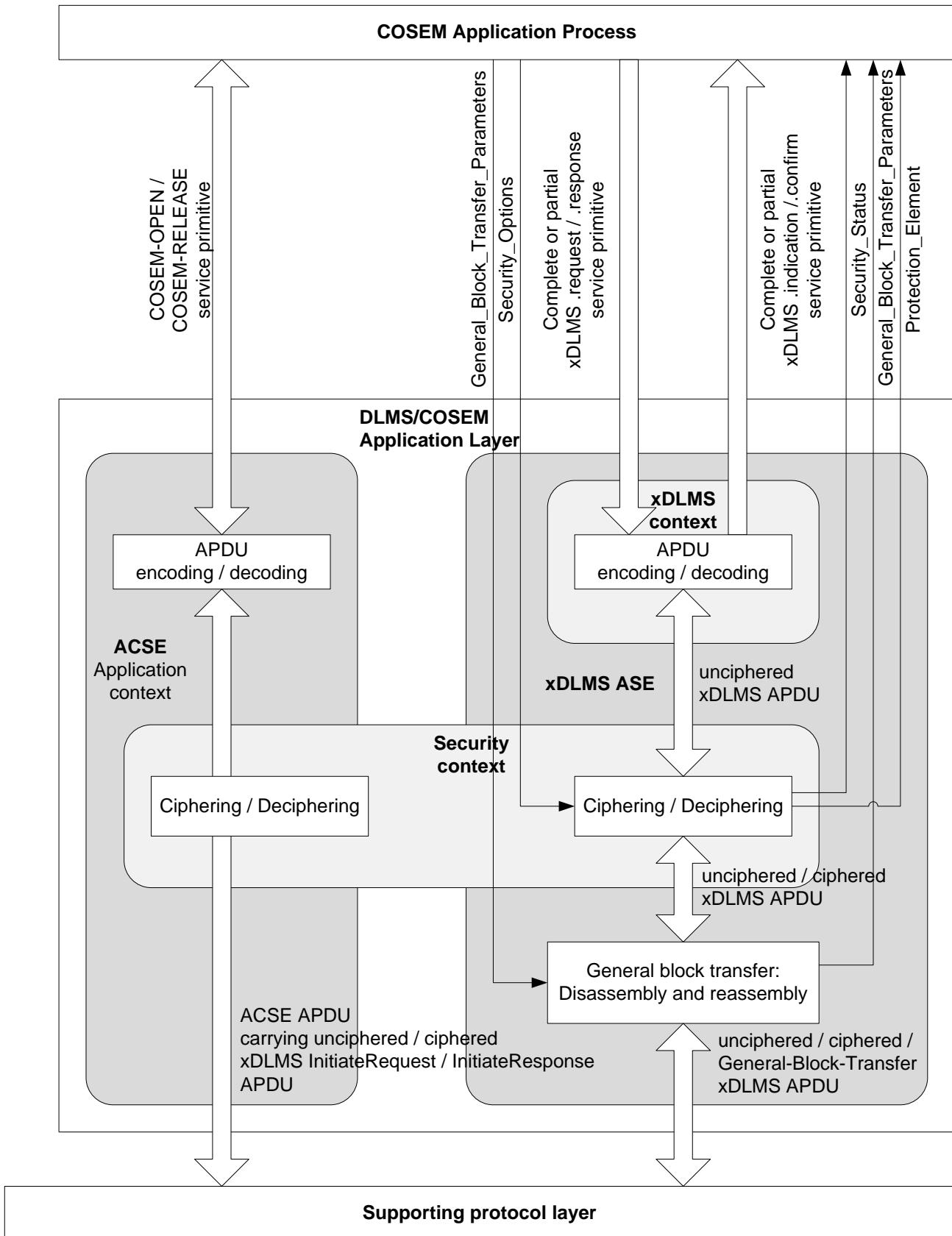
Use

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. The origin of such an event can be an external event (for example the physical line is broken), or an action of a supporting layer connection manager AP, present in some profiles, when the supporting 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 88 and Table 49.



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 88 – Additional service parameters to control cryptographic protection and GBT

Table 49 – 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 50, 9.4.6.13 and Figure 111.

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 50.

The Protection_Element parameter is conditional: it is present only if the APDU has been authenticated or signed. See Table 50.

Table 50 – Security parameters

	.request	.indication	.response	.confirm
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 General_Ded_Ciphering</i>				
System_Title	U	–	U (=)	–
<i>With General_Ciphering and 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 (=)	–

Table 50 – Security parameters (continued)

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 General_Ded_Ciphering</i>				
System_Title	-	U	-	U (=)
<i>With General_Ciphering and 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 51;
- 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 51 – 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 36.	

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;

NOTE 3 If any of the four parameters above is not used, than an octet-string of length zero shall be included.

- 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;

The Key_Info_Options parameter is mandatory: 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 37.

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

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.

Semantics

The GET service primitives shall provide parameters as shown in Table 52.

Table 52 – 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 50.				

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 53.

Table 53 – 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);
NOTE 2 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.

Use

Possible logical sequences of the GET service primitives are illustrated in Figure 87:

- 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

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.

Semantics

The SET service primitives shall provide parameters as shown in Table 54.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	238/500
-----------------------	------------	------------------------	---------

Table 54 – 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 50.				

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 55.

Table 55 – SET service request and response types

Request type		Response type	
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 list of results, either the success to write or a reason for failing to write the attribute (Data_Access_Result), for each public attribute, in the order of their appearance in the given object specification.

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.

Use

Possible logical sequences of the SET service primitives is illustrated in Figure 87:

- 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

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.)

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	241/500
-----------------------	------------	------------------------	---------

Semantics

The ACTION service primitives shall provide parameters as shown in Table 56.

Table 56 – 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 50.				

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 57.

Table 57 – 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.

Use

Possible logical sequences of the ACTION service primitives are illustrated in Figure 87:

- 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).

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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	244/500
-----------------------	------------	------------------------	---------

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 Invoke_Id parameter of the GET, SET and ACTION services allows the client and the server to pair requests and responses. The range of the Invoke_Id is 0...15.

In some cases this is not sufficient. To support those cases, the ACCESS service uses a Long_Invoke_Id parameter. The range of the Long_Invoke_Id is 0...16 777 215.

NOTE Description of the circumstances when long Invoke_id-s are useful is beyond the Scope of this Technical Report.

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

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:

- 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.

Semantics

The ACCESS service primitives shall provide parameters as shown in Table 58.

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 58 – 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 (=) ¹	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 (=)

¹ 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;

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	249/500
-----------------------	------------	------------------------	---------

- 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.

Use

Possible logical sequences of the ACCESS service primitives are illustrated in Figure 87:

- 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 or a list of results indicating the reason for the failure.

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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	250/500
-----------------------	------------	------------------------	---------

9.3.10 The DataNotification service

Function

The DataNotification service is an unsolicited, unconfirmed service. It is used by the server to push data to the client. It is an unconfirmed service. The push process is configured by Push setup objects; see DLMS UA 1000-1 Ed. 12.1:2015 4.4.8.

Semantics

The DataNotification service primitives shall provide parameters as shown in Table 59.

Table 59 – Service parameters of the DataNotification service primitives

	.request	.indication
Long_Invoke_Id	M	M (=)
Self_Descriptive	–	–
Processing_Option	–	–
Service_Class	–	–
Priority	M	M (=)
Date_Time	U	U (=)
Notification_Body	M	M (=)

The Long_Invoke_Id parameter identifies the instance of the service invocation. See also 9.3.9.

The Self_Descriptive, Processing_Option and Service_Class parameters are not used in the case of this service.

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 service primitive is invoked.

The Notification_Body parameter contains the push data.

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.

Use

Possible logical sequences of the DataNotification service primitives are illustrated in Figure 87 f) and g).

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 protocol for the DataNotification service is specified in 9.4.6.7.

9.3.11 The EventNotification service

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.

Semantics

The EventNotification service primitives shall provide parameters as shown in Table 60.

Table 60 – 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.

Use

A possible logical sequence of the EventNotification service primitives is illustrated in Figure 87 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

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.

Semantics

The TriggerEventNotificationSending.request service primitive shall provide parameters as shown in Table 61.

Table 61 – Service parameters of the TriggerEventNotificationSending.request service primitive

Protocol_Parameters	.request
	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.

Use

Upon the reception of a TriggerEventNotificationSending.request service invocation from the client AP, the client AL shall invoke the corresponding supporting 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 62:

- 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 62 – 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

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.

Semantics

The Read service primitives shall provide service parameters as shown in Table 63.

Table 63 – 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 (=)		
Block_Number_Access	S	S (=)		
Block_Number	M	M (=)		
Result (+)			S	S (=)
Read_Result { Read_Result }	–	–	M	M (=)
Data			S	S (=)
Data_Access_Error			S	S (=)
Data_Block_Result			S	S (=)
Last_Block			M	M (=)
Block_Number			M	M (=)
Raw_Data			M	M (=)
Block_Number			S	S (=)
Result (–)	–	–	S	S (=)
Error_Type			M	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 64.

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 64 – 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 ¹ 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 ¹ of COSEM object attributes to be read selectively.	Data {Data}	As above.
		Data_Access_Error {Data_Access_Error}	
		Data_Block_Result	Delivers the method invocation return parameters. NOTE If parameters are returned, this implies that the method invocation succeeded.
Read_Data_Block_Access	References a list ¹ of COSEM object methods, with method invocation parameters.	Data {Data}	Data_Access_Error {Data_Access_Error}
		Data_Block_Result	Provides the reason for the method invocation to fail.
		Data_Block_Result	As above.
Block_Number_Access	Carries the number of the latest data block received.	Block_Number	Carries the number of the latest data block received.
NOTE The same Read.response choice may be present more than once, to show the possible responses to each request.			
¹ 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;

NOTEWith 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);

NOTE 2 If there are no return parameters, Data should be null data. 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:

- 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.

Use

A possible logical sequence of the Read service primitives is illustrated in Figure 87 item a).

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	256/500
-----------------------	------------	------------------------	---------

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

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.

Semantics

The Write service primitives shall provide service parameters as shown in Table 65.

Table 65 – Service parameters of the Write service

	.request	.indication	.response	.confirm
Variable_Access_Specification { Variable_Access_Specification } Variable_Name Parameterized_Access Variable_Name Selector Parameter	M S S M M M	M(=) S (=) S (=) M (=) M (=) M (=)	–	–
Write_Data_Block_Access Last_Block Block_Number	S M M	S (=) M (=) M (=)	–	–
Data { Data }	M	M (=)	–	–
Result (+)	–	–	S	S (=)
Write_Result { Write_Result } Success Data_Access_Error	-	-	S S S	S (=) S (=) S (=)
Block_Number			S	S (=)
Result (-)			S	S (=)

	.request	.indication	.response	.confirm
Error_Type			M	M (=)

NOTE For security parameters, see Table 50.

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 66. 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 66 – 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 ¹ 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.
	References a list ¹ of COSEM object attributes to be written selectively. The Data service parameter carries the data to be written.	Data_Access_Error {Data_Access_Error}	Provides the reason for the write to fail.
Parameterized_Access {Parameterized_Access}	Carries block transfer control information. The Data service parameter carries raw-data, including the encoded form of the list ¹ of COSEM object attribute or method references, and the list of data to be written or the list of method invocation parameters.	Success {Success}	As above.
		Data_Access_Error {Data_Access_Error}	
Write_Data_Block_Access		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.

¹ 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.

Use

A possible logical sequence of the Write service primitives is illustrated in Figure 87 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

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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	259/500
-----------------------	------------	------------------------	---------

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.

Semantics

The UnconfirmedWrite service primitives shall provide service parameters as shown in Table 67.

Table 67 – 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 50.		

The use of the different variants of the Variable-Access-Specification service parameter of the UnconfirmedWrite.request service primitive is shown in Table 68. 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 68 – 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. 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.

Use

A possible logical sequence of the Write service primitives is illustrated in Figure 87 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

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.

Semantics

The InformationReport service primitives shall provide parameters as shown in Table 69.

Table 69 – Service parameters of the InformationReport service

	.request	.indication
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

Function

The function of the SetMapperTable service is to manage the Client SN_Mapper ASE.

Semantics

There is only one primitive, the .request primitive. It shall provide parameters as follows:

Table 70 – 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.

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 71 and Table 72 provide a summary of the DLMS/COSEM application layer services.

Table 71 – 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 72 – Summary of xDLMS services

Client side	Server side
LN referencing	
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
SN referencing	
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
NOTE The DataNotification service can be used in application contexts using either SN referencing or LN referencing.	

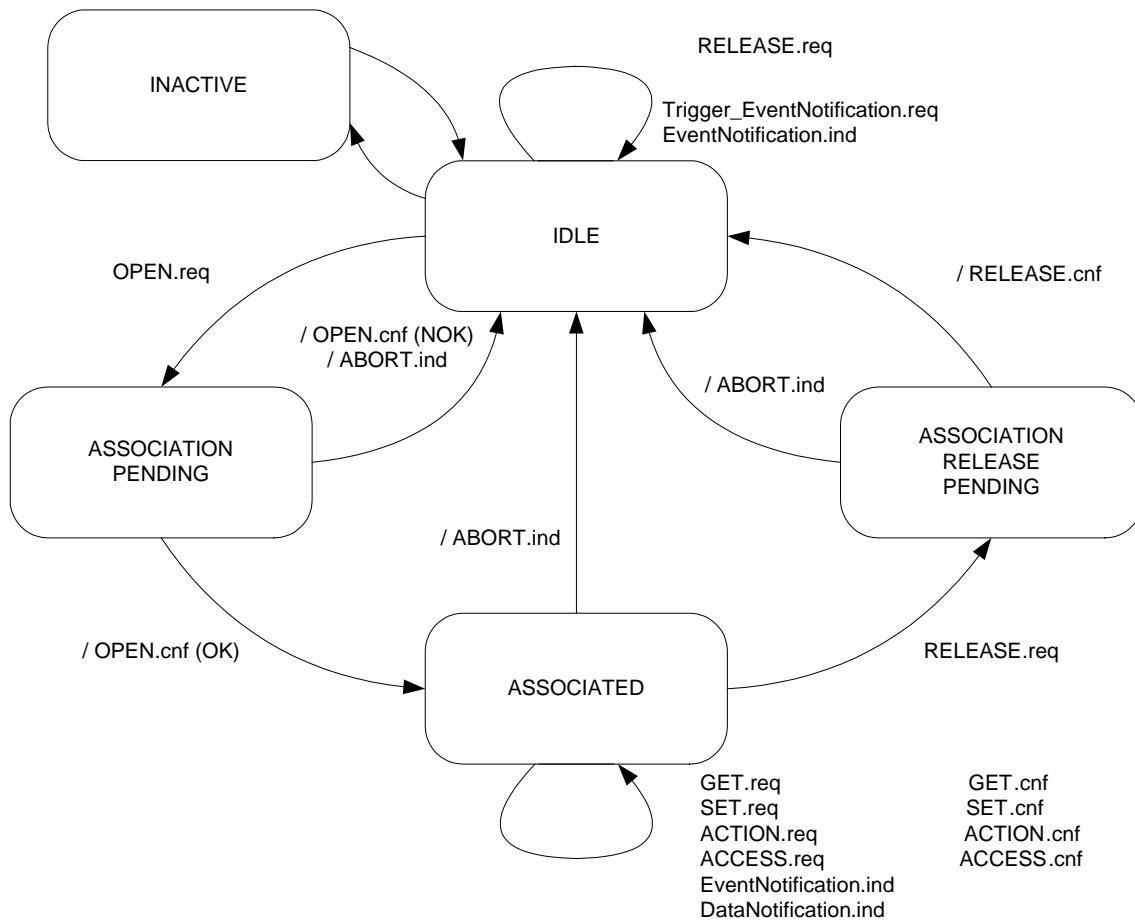
When the server uses SN referencing, a mapping between the service primitives using LN referencing and SN referencing takes place on the client side. This mapping is specified in 9.4.6.9, 9.4.6.10, 9.4.6.11 and 9.4.6.12.

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 89 shows the state machine for the client side CF, see also Figure 61.



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 89 – 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 ⁸ . 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.
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,

⁸ 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.

(/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 90 shows the state machine for the server side CF, see Figure 61.

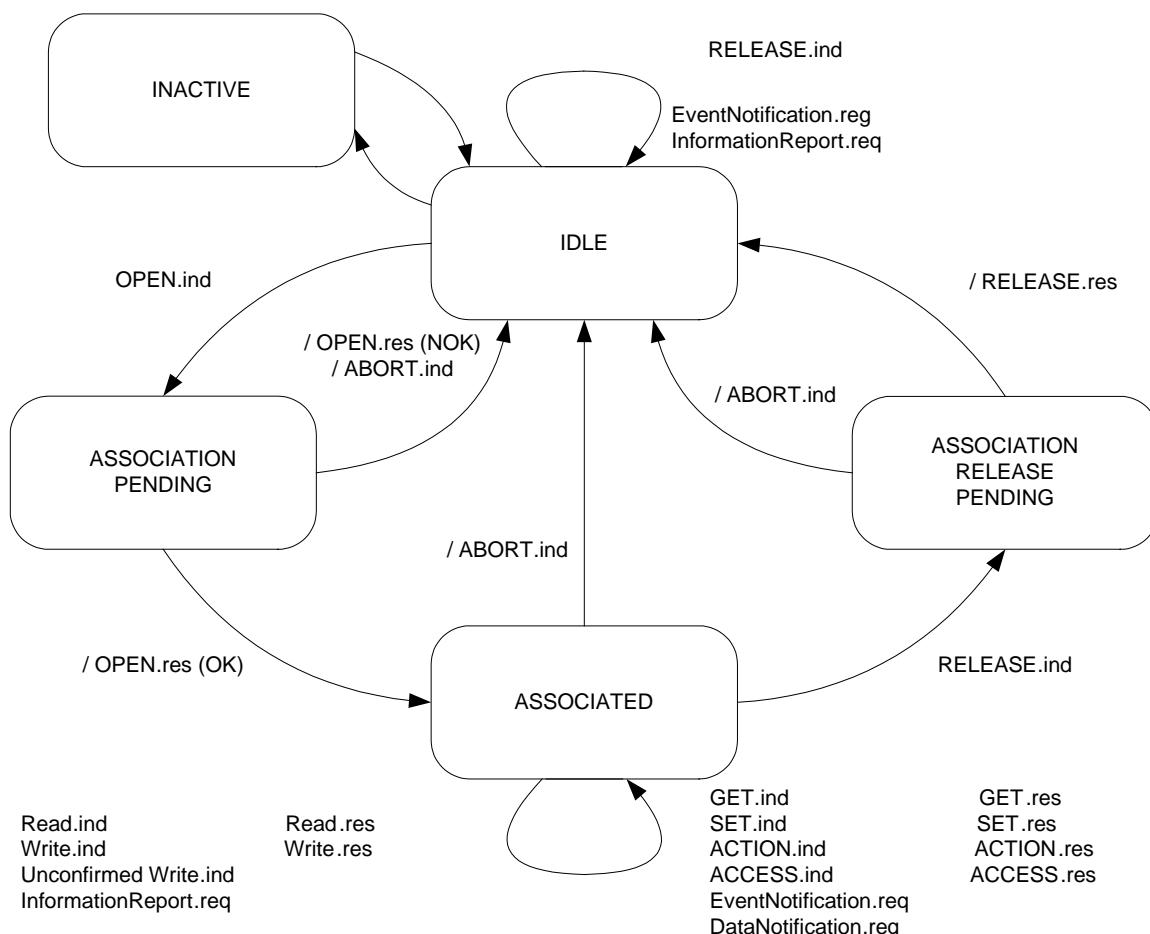


Figure 90 – Partial state machine for the server side control function

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 ⁹ . 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;
- Higher Level Association functional unit; and
- Nested Association 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.

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.

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 73 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.

⁹ 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.

Table 73 – 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 ²⁾ (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 ³⁾ (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 M
	A-RELEASE	RLRQ	reason user-information	U U
		RLRE	reason user-information	U U
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 ACPM 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"> - 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; - 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; - 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. 	

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 uses "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.

NOTE 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 74:

Table 74 – 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 9.2.2.2.2.2;
- 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 75 – 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 76. See also Table 18.

Table 76 – 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). 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.

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.

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 91 gives the MSC for the case, when:

- the COSEM-OPEN.request primitive requests a confirmed AA;
- the connection of the supporting 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 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 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 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.

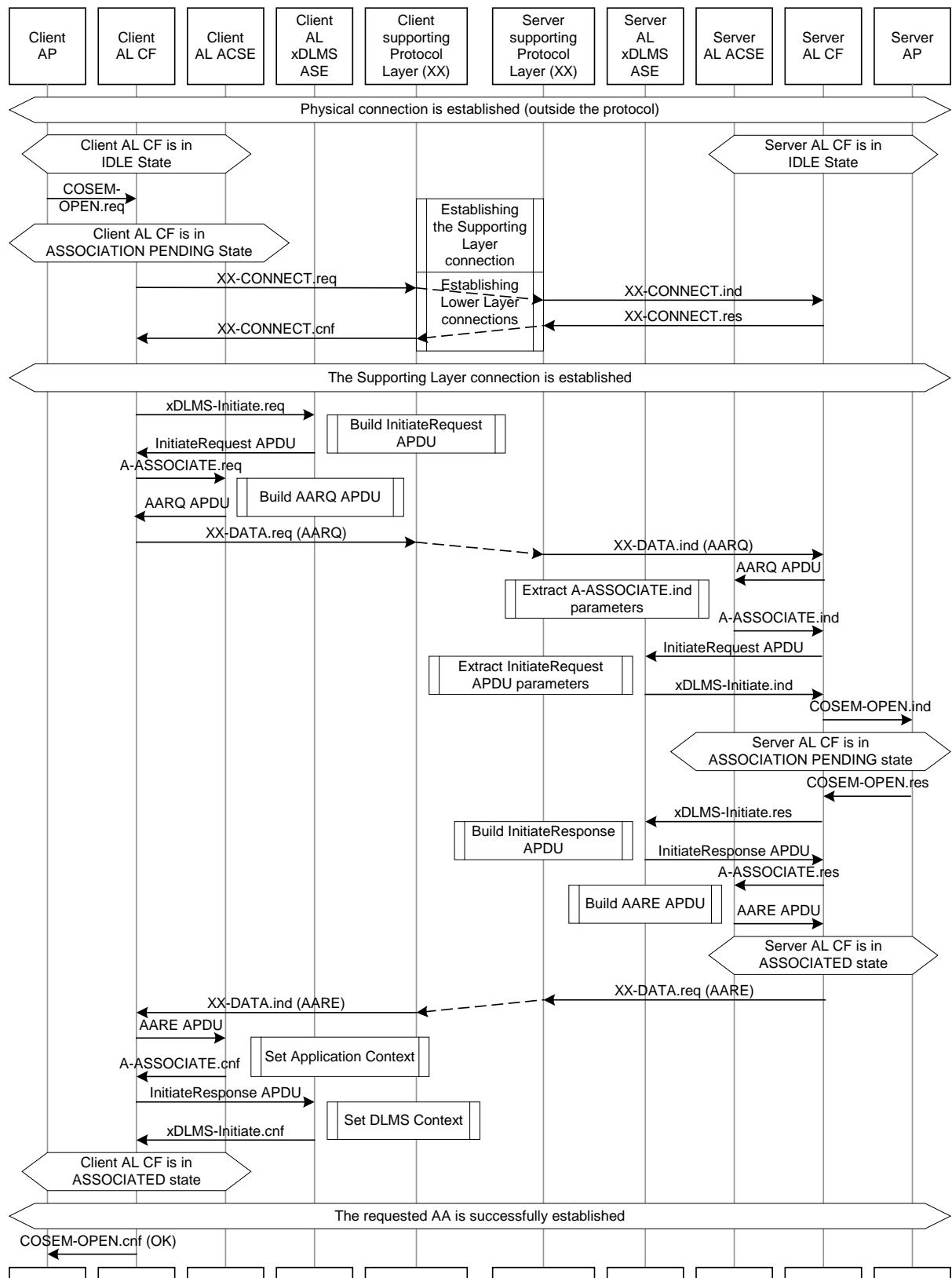


Figure 91 – MSC for successful AA establishment preceded by a successful lower layer connection establishment

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 Ed. 12.1:2015. 4.4.2;
- 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.

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:
 - c) 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;

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	273/500
-----------------------	------------	------------------------	---------

- d) 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 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	274/500
-----------------------	------------	------------------------	---------

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.

9.4.5.2 Graceful release of an application association

DLMS/COSEM provides two mechanisms to release AAs:

- by disconnecting the supporting layer of the AL;
- by using the ACSE A-Release service.

The first mechanism shall be supported in all profiles where the supporting layer of the AL is connection oriented.

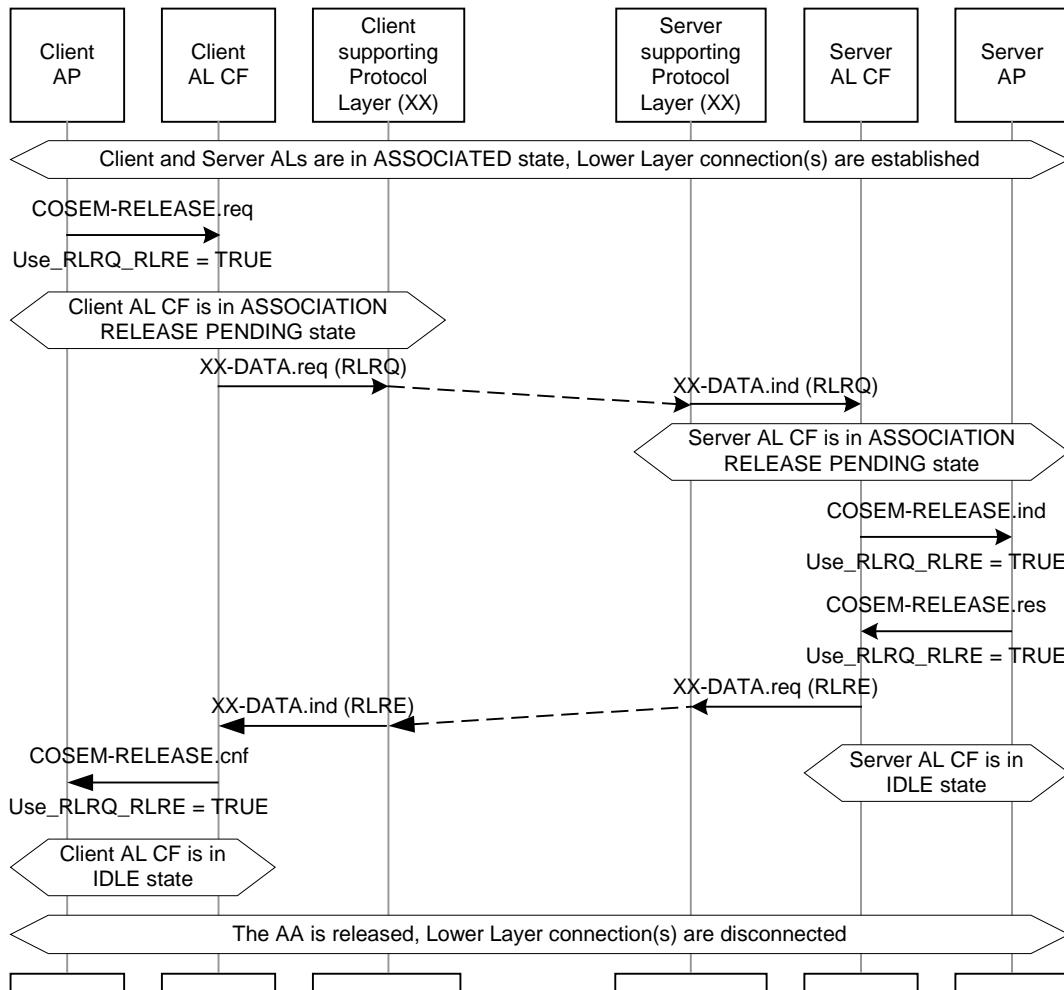
EXAMPLE The 3-layer, HDLC based, connection oriented profile.

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 layer shall release all AAs built on that supporting layer connection.

The second mechanism can be used to release an AA without disconnecting the supporting layer. It shall be supported in all profiles when the supporting layer is connectionless. It may be also used when the supporting layer is connection-oriented but the connection is not managed by the AL, or disconnection of the supporting 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 92.



NOTE The release of an AA may require internal communication between the ASEs of the CF. These are not shown in Figure 92 – Figure 94.

Figure 92 – 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 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 93 gives an example of graceful releasing a confirmed AA by disconnecting the corresponding lower layer connection.

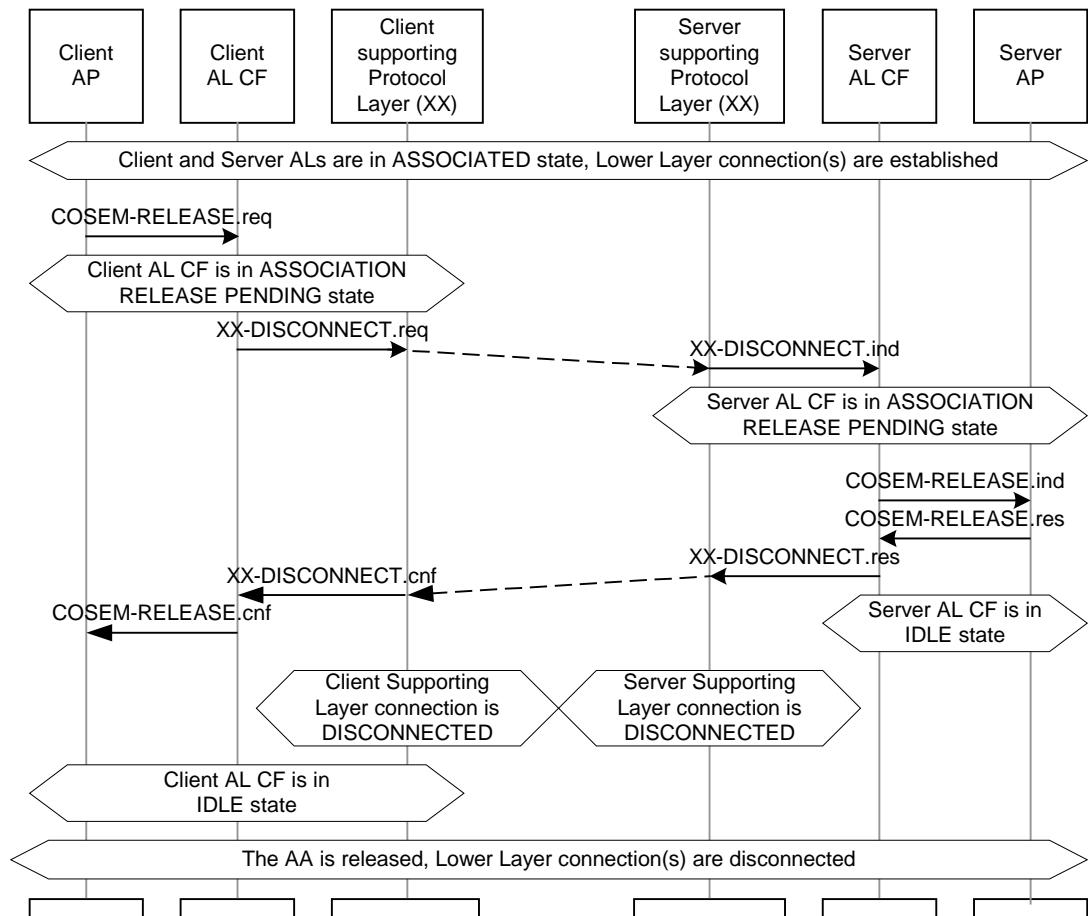


Figure 93 – Graceful AA release by disconnecting the supporting layer

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 ASSOCATION 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 to the server.

When the server AL CF receives the XX-DISCONNECT.request primitive, the CF enters the ASSOCATION 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 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 94 shows the message sequence chart for aborting the AA, due to the detection of a physical disconnection.

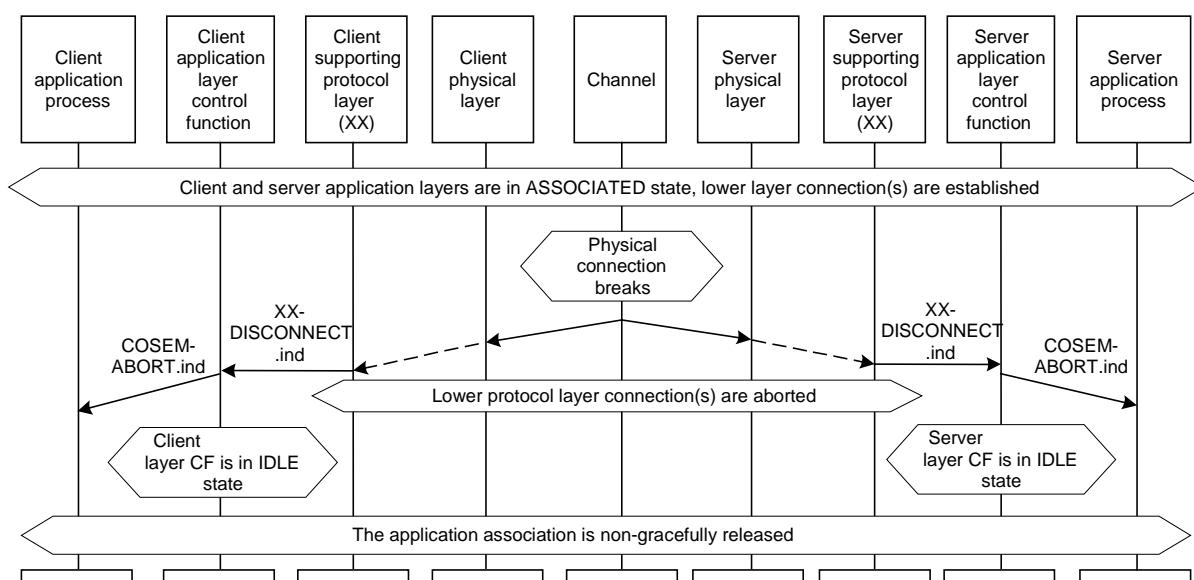


Figure 94 – Aborting an AA following a PH-ABORT.indication

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 77.

Table 77 – xDLMS Conformance block

Conformance block bit	Reserved	LN referencing	SN referencing
0	x		
1		general-protection ¹	general-protection ¹
2		general-block-transfer	general-block-transfer
3			read
4			write
5			unconfirmed-write
6	x		
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	

¹ general-protection includes general-glo-ciphering, general-ded-ciphering, general-ciphering and general-signing.

9.4.6.2 Confirmed and unconfirmed xDLMS service invocations

In general, xDLMS services may be invoked in a confirmed or an unconfirmed manner. The time sequence of the service primitives corresponds to:

- Figure 87 item a) in the case of confirmed service invocations; and
- Figure 87 item d) in the case of unconfirmed service invocations.

A client AP that desires to access 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.

The server AP, upon the receipt of the .indication primitive, checks whether the service can be provided or not (validity, client access rights, availability, etc.). If everything is OK, 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 server. 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).

9.4.6.3 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.

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 1 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 78.

Table 78 – 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 95 shows the MSC for a confirmed GET service in the case of success, without block transfer.

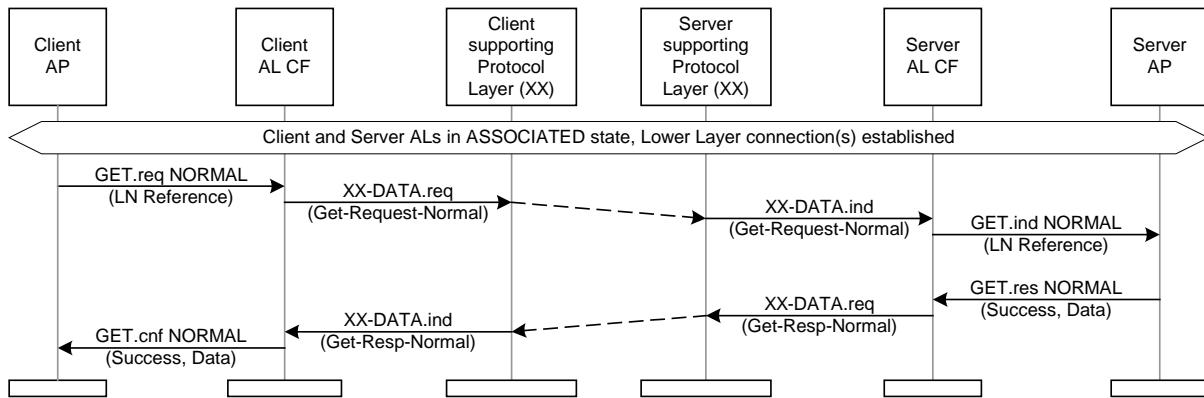
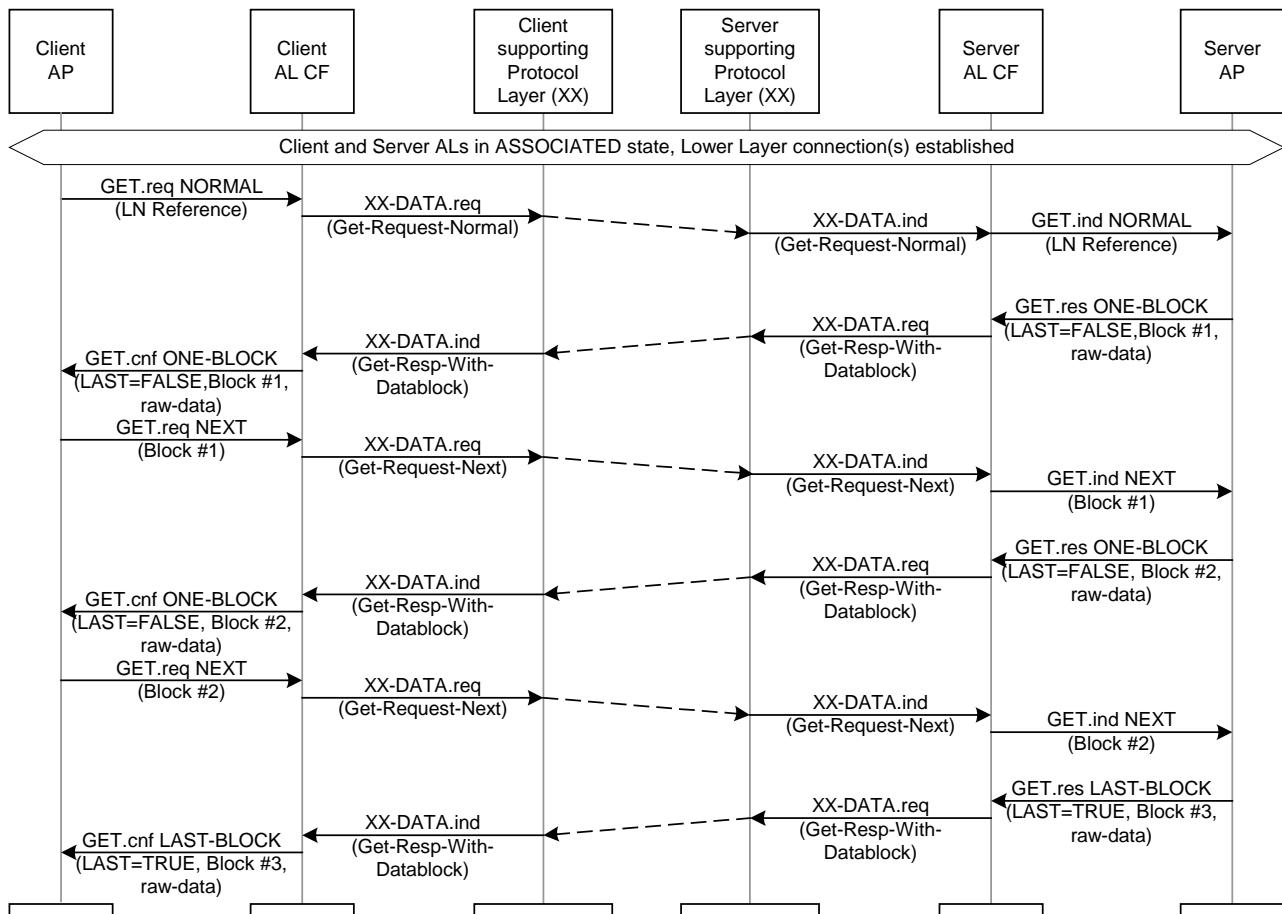
**Figure 95 – MSC of the GET service**

Figure 96 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 96 – 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 DataAccessResult;

- 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 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	282/500
-----------------------	------------	------------------------	---------

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 97:

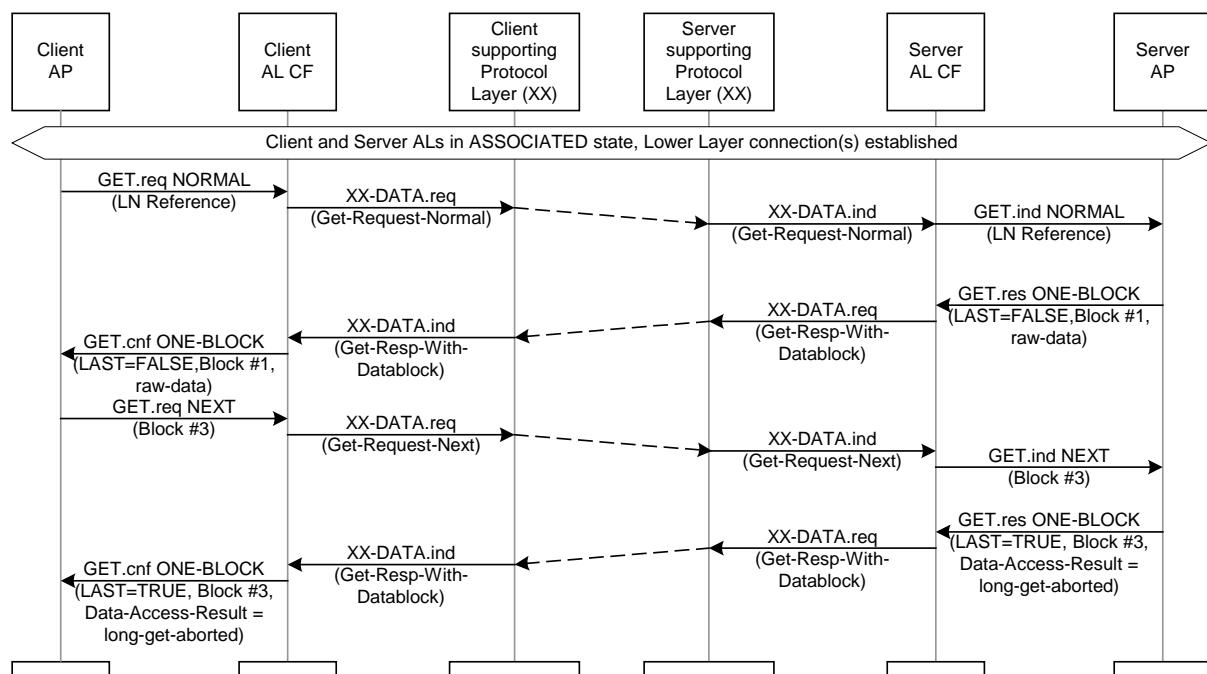


Figure 97 – MSC of the GET service with block transfer, long GET aborted

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 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 79.

Table 79 – 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		
LAST-BLOCK	Set-Request-With-Datablock Last-Block = TRUE	Set-Response-Last-Datablock	LAST-BLOCK 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 98 shows the MSC of a confirmed SET service, in the case of success, without block transfer.

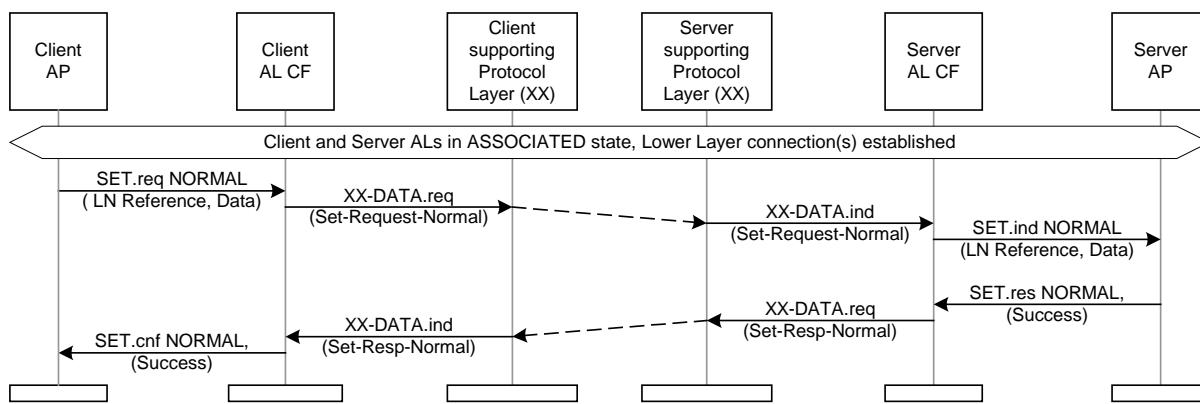
**Figure 98 – MSC of the SET service**

Figure 99 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.

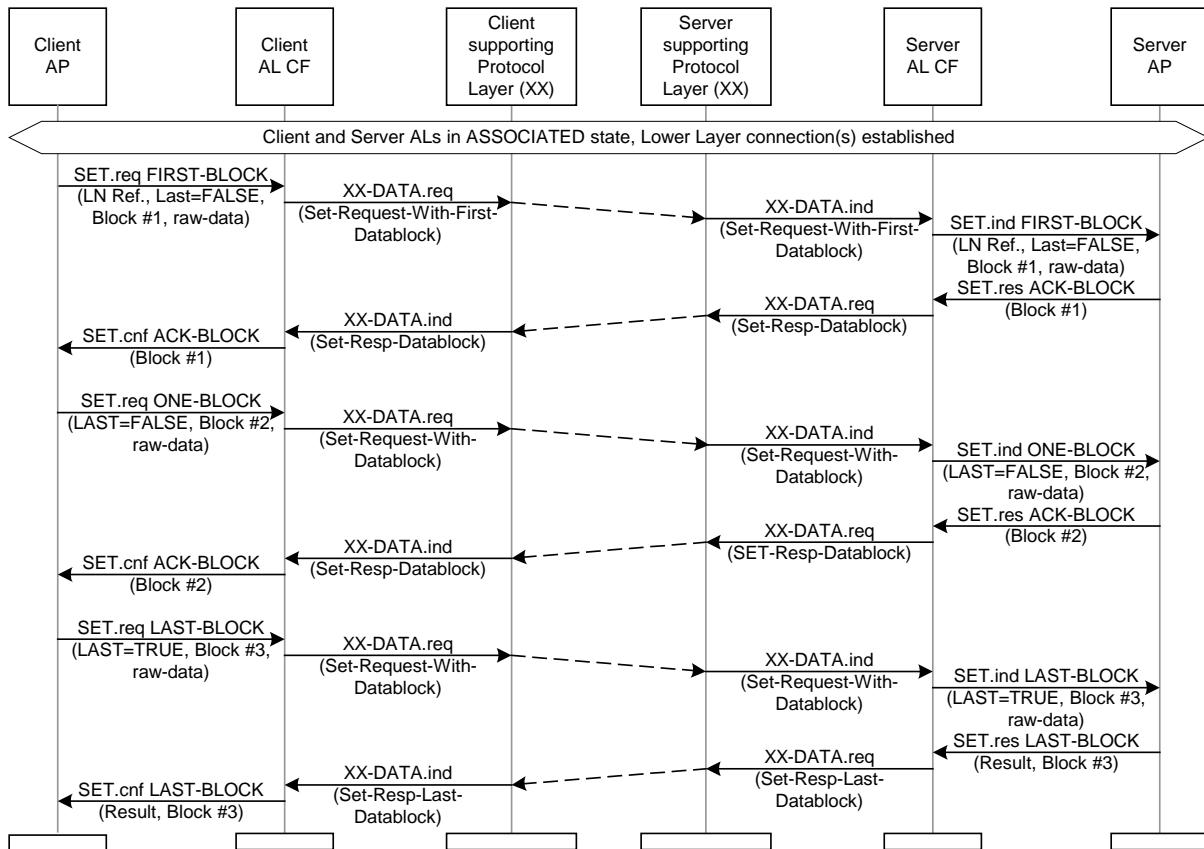


Figure 99 – MSC of the SET service with block transfer

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: $B_1, B_2, B_3, \dots, B_K$.

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 $B_{K+1}, B_{K+2}, B_{K+3}, \dots, B_L$, 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 $B_M, B_{M+1}, B_{M+2}, \dots, B_N$ 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.

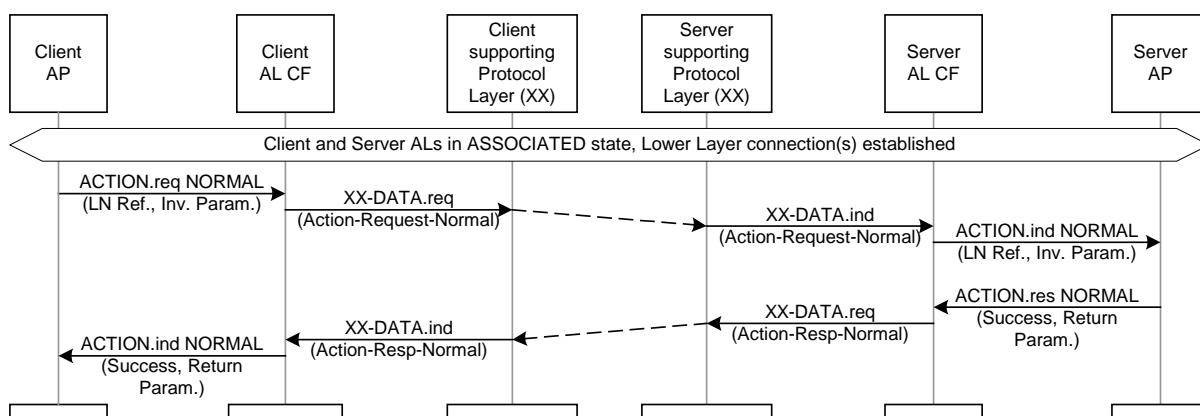
NOTE In some DLMS/COSEM communication profiles segmentation is available to transfer long APDUs.

The ACTION service primitive types and the corresponding APDUs are shown in Table 80.

Table 80 – ACTION service types and APDUs

SET .req / .ind	Request APDU	Response APDU	SET .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-First-Pblock	Action-Response-Next-Pblock	NEXT

Figure 100 illustrates the MSC of a confirmed ACTION service in the case of success, without block transfer.

**Figure 100 – 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 101 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.

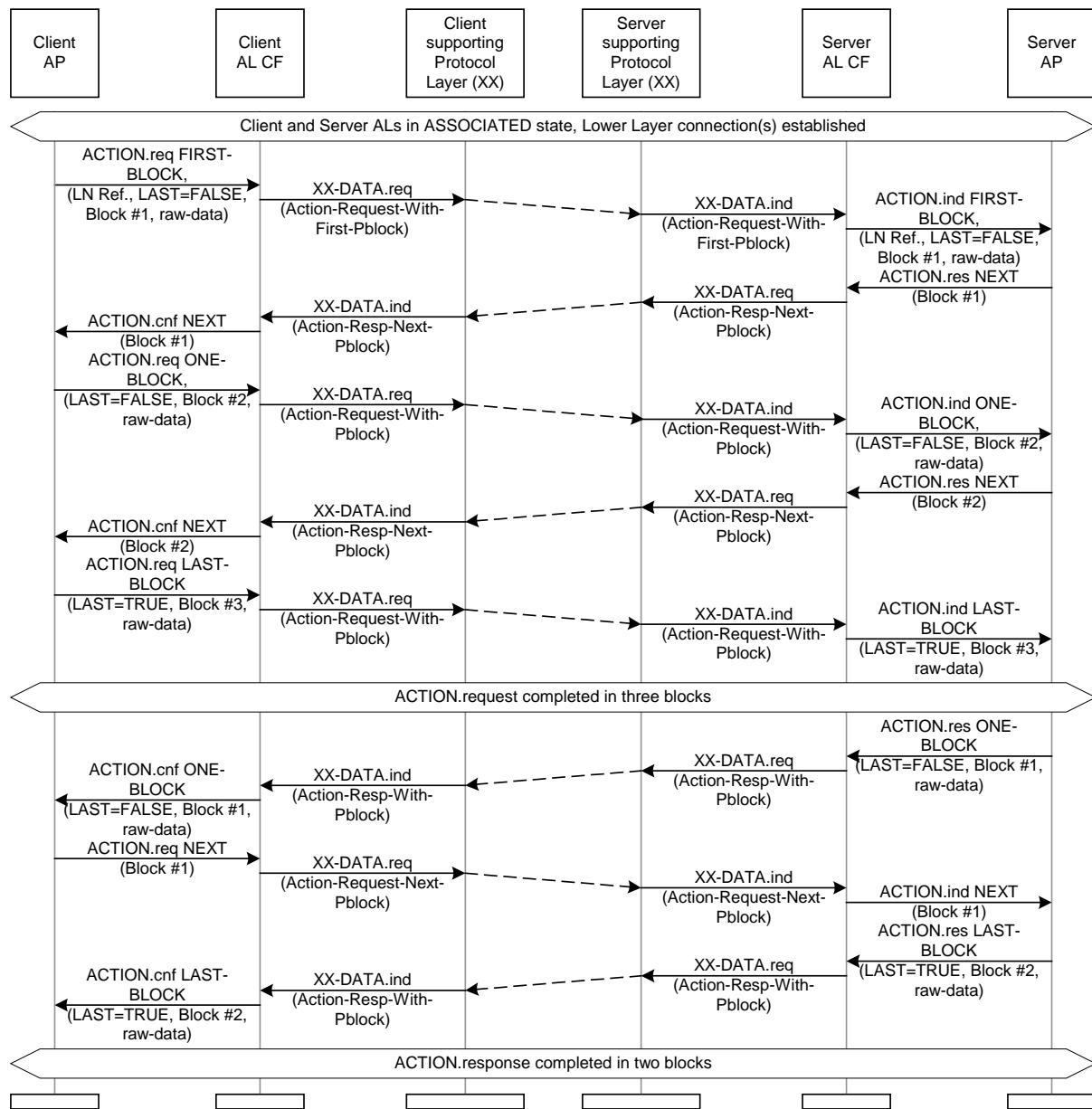


Figure 101 – MSC of the ACTION service with block transfer

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 102 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.

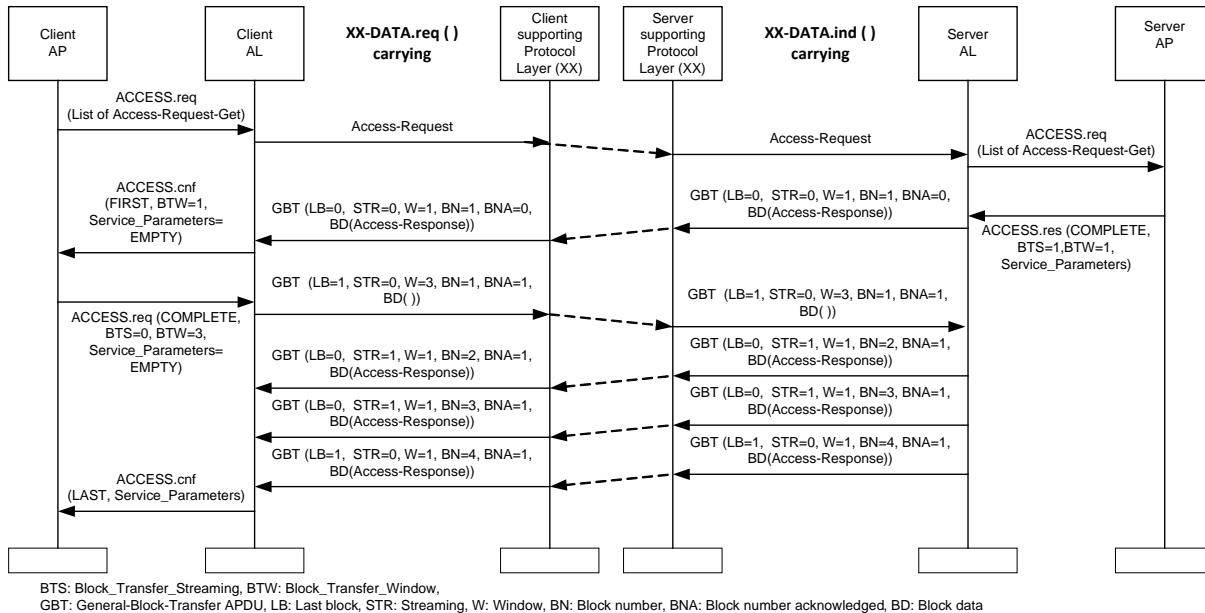
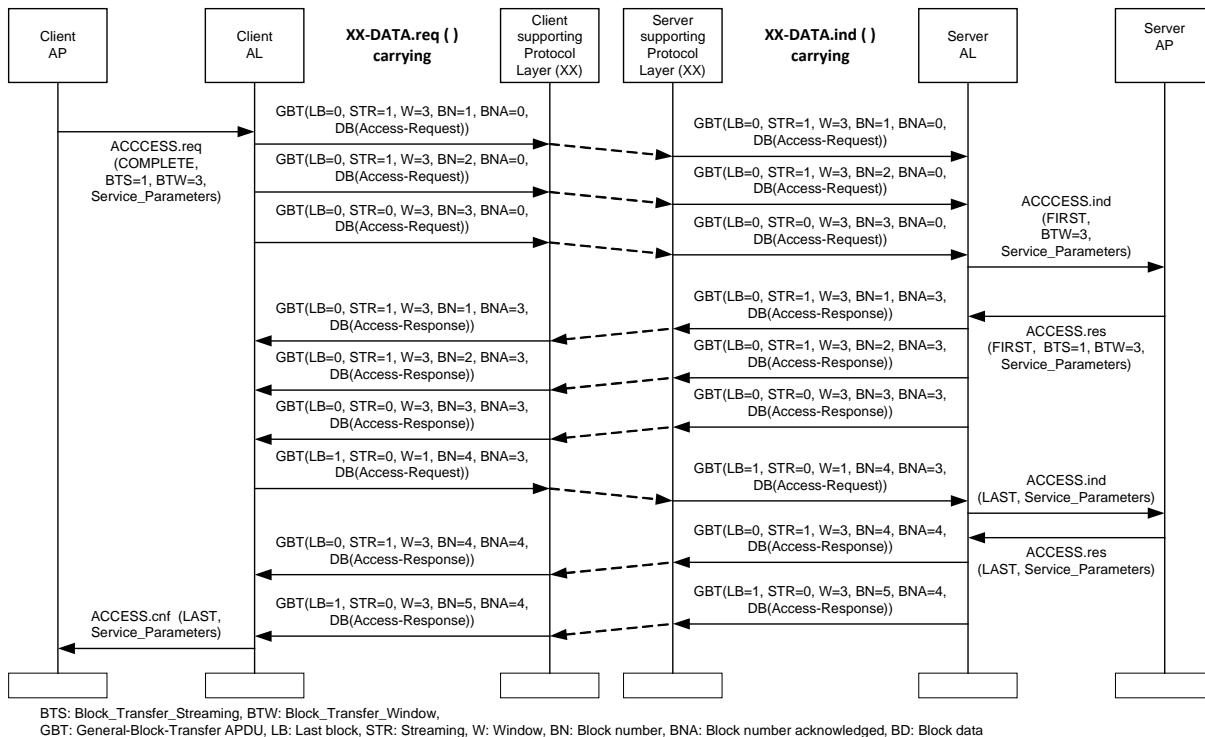
**Figure 102 – Access Service with long response**

Figure 103 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 103 – 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 sends it to the client.

When the client AL receives this APDU, it invokes the DataNotification.indication service primitive.

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 service-specific or the general block transfer mechanism may be used.

See also Figure 118.

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 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 81;
- 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 82.

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 81 – Mapping between the GET and the Read service

From GET.request of type	To Read.request	SN APDU
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})
To GET.confirm of type	SN APDU	From Read.response
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 82 – Mapping between the ACTION and the Read service

From ACTION.request of type	To Read.request	SN APDU
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)
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)

From ACTION.request of type	To Read.request	SN APDU
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)
To ACTION.confirm	SN APDU	From Read.response
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 104 shows the MSC of a Read service used to read the value of a single attribute.

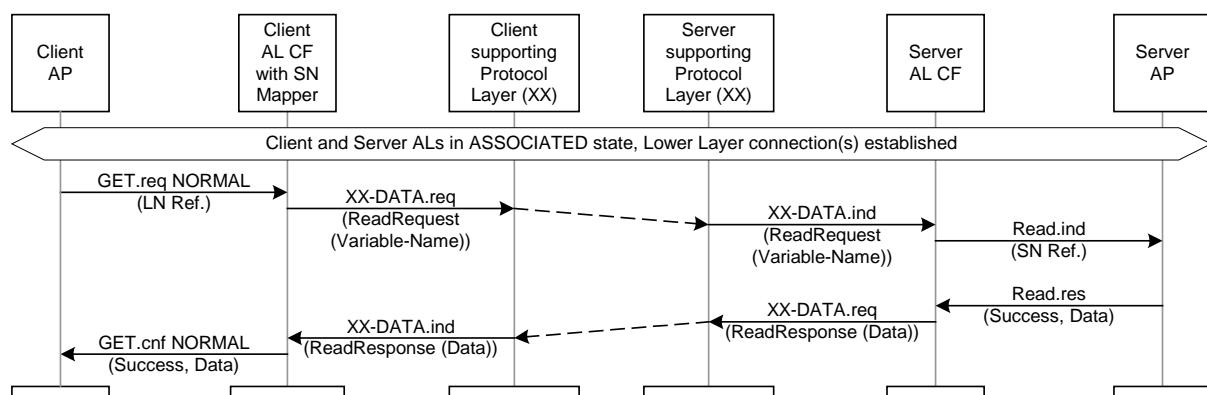


Figure 104 – MSC of the Read service used for reading an attribute

Figure 105 shows the MSC of a Read service used to invoke a single method.

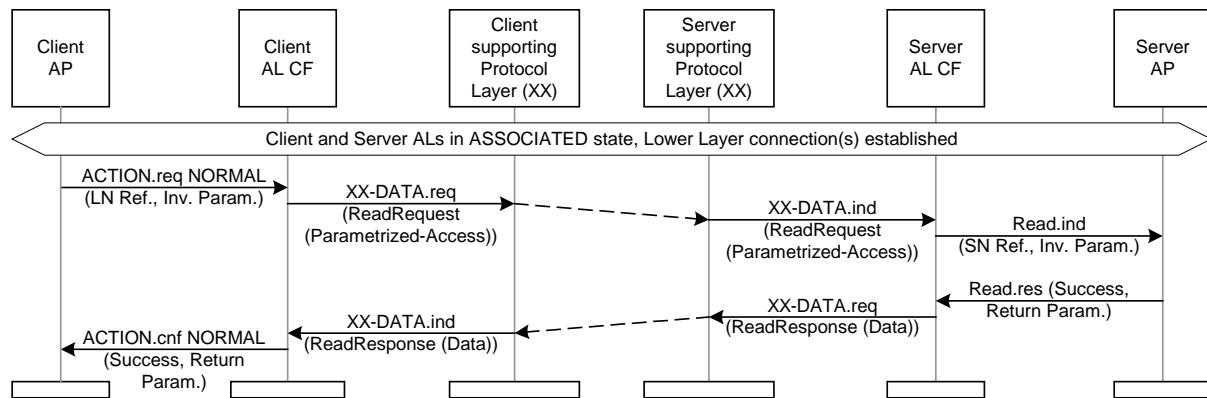


Figure 105 – MSC of the Read service used for invoking a method

Figure 106 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.

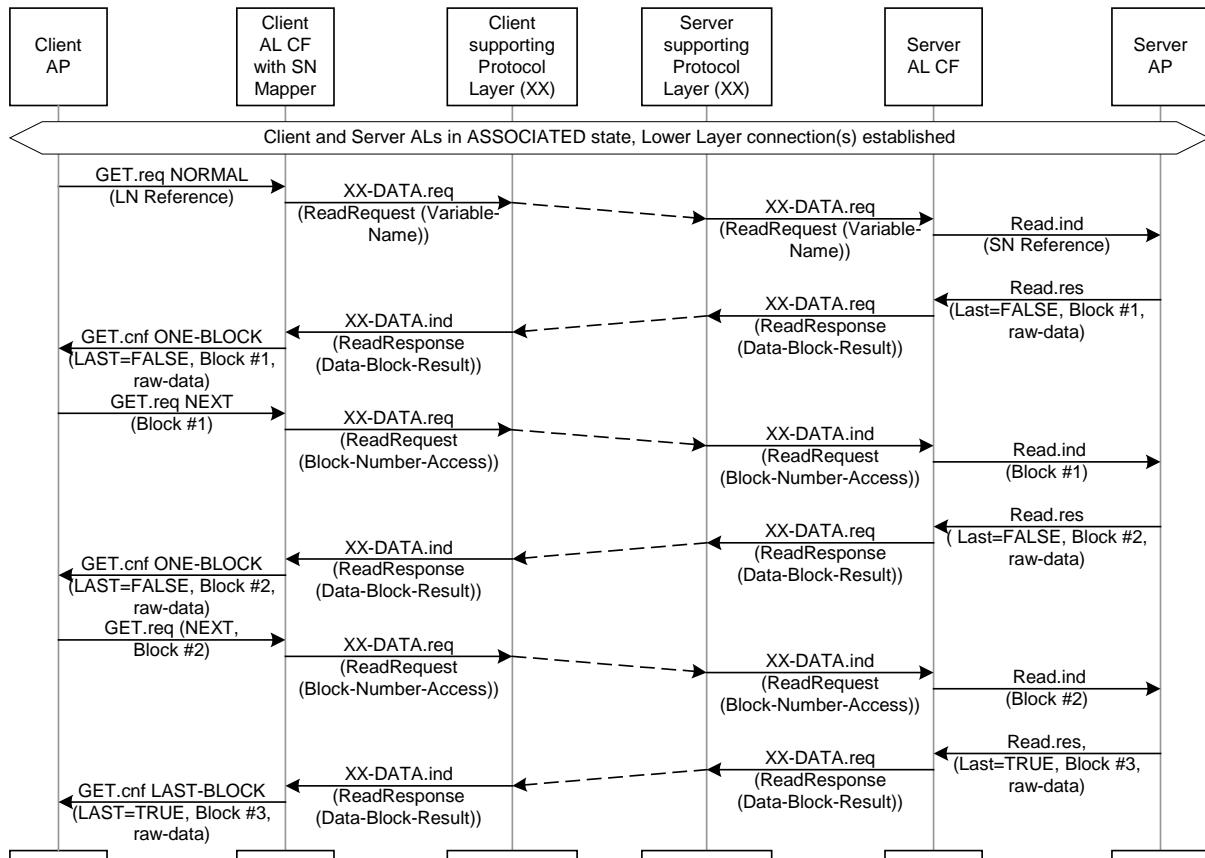


Figure 106 – MSC of the Read Service used for reading an attribute, with block transfer

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 83;
- 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 84.

Table 83 – Mapping between the SET and the Write service

From SET.request of type	To Write.request	SN APDU
NORMAL	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name Parameterized_Access;	WriteRequest ::= (variable-name 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 Parameterized_Access;	WriteRequest ::= ({variable-name parameterized-access}, {data})
FIRST-BLOCK-WITH-LIST	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification =	WriteRequest ::= (write-data-block-access, data)

From SET.request of type	To Write.request	SN APDU
	Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1 Data = as above	
To SET.confirm of type	SN APDU	From Write.response
NORMAL	WriteResponse ::= (success data-access-error)	Write.response (Write_Result) Write_Result = Success Data_Access_Error;
ACK-BLOCK	WriteResponse ::= (block-number)	Write.response (Write_Result) Write_Result = Block_Number;
LAST-BLOCK	WriteResponse ::= (success data-access-error)	Write.response (Write_Result) Write_Result = Success Data_Access_Error;
WITH-LIST	WriteResponse ::= ({success data-access-error})	Write.response ({Write_Result}) Write_Result = Success Data_Access_Error;
LAST-BLOCK-WITH-LIST	WriteResponse ::= ({success data-access-error})	Write.response ({Write_Result}) Write_Result = Success Data_Access_Error;

Table 84 – Mapping between the ACTION and the Write service

From ACTION.request of type	To Write.request	SN APDU
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)

From ACTION.request of type	To Write.request	SN APDU
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)
To ACTION.confirm	SN APDU	From Write.response
NORMAL	WriteResponse ::= (success data-access-error)	Write.response (Write_Result) Write_Result = Success Data_Access_Error
NEXT	WriteResponse ::= (block-number)	Write.response (Block_Number)
To ACTION.confirm	SN APDU	From Write.response
WITH-LIST	WriteResponse ::= ({success data-access-error})	Write.response ({Write_Result}) Write_Result = Success Data_Access_Error

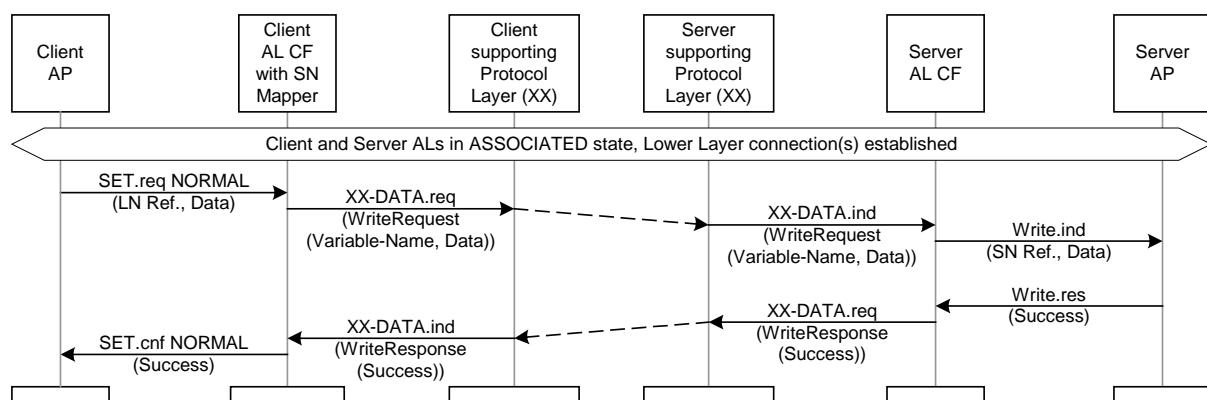
**Figure 107 – MSC of the Write service used for writing an attribute**

Figure 107 shows the MSC of a Write service used to write the value of a single attribute, in the case of success.

Figure 108 shows the MSC of a Write service used to invoke a single method, in the case of success.

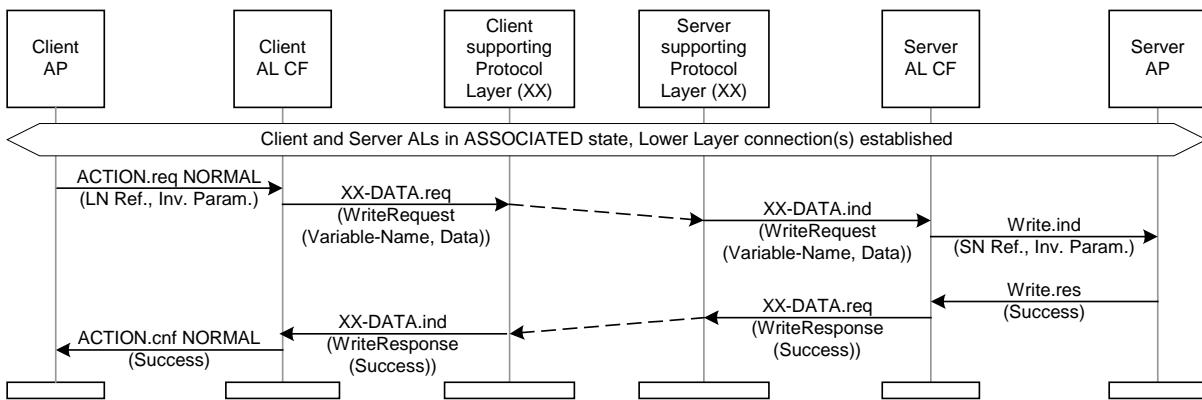
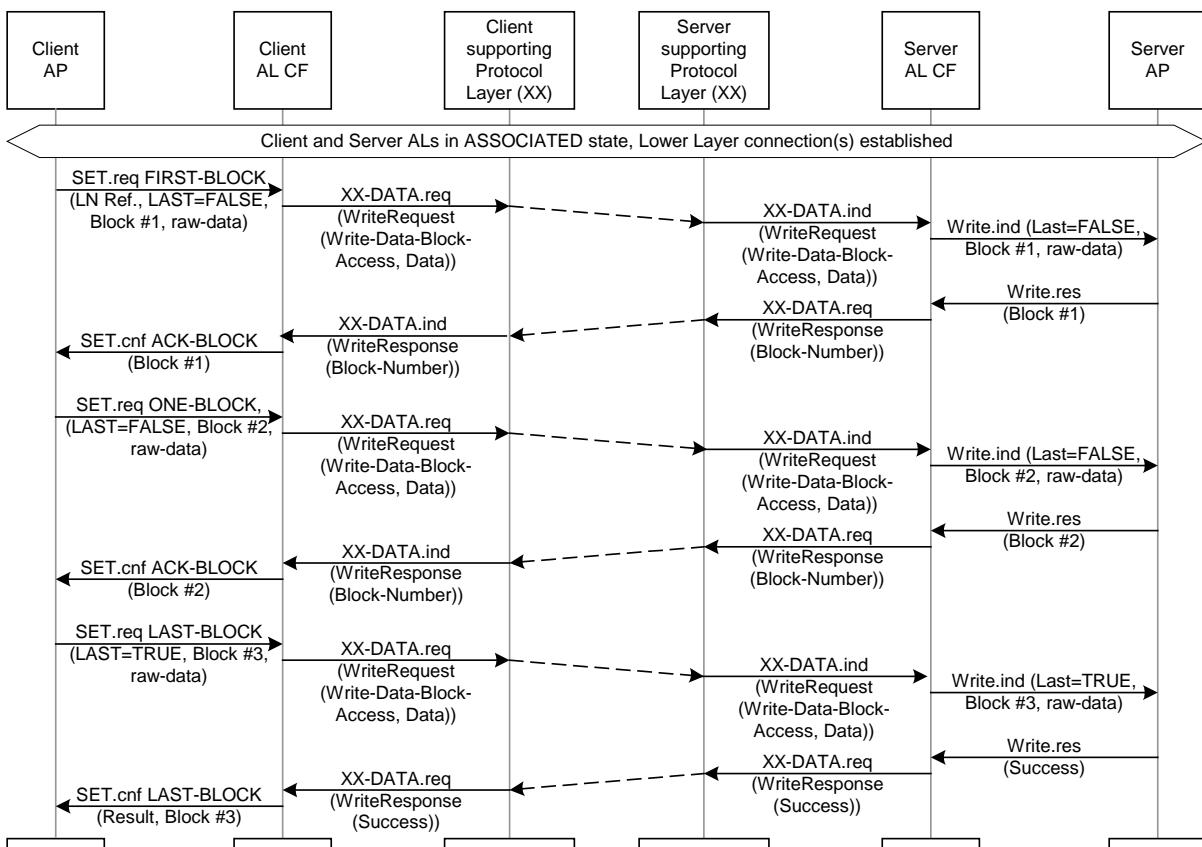
**Figure 108 – MSC of the Write service used for invoking a method**

Figure 109 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 109 – 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 85;
- 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 86.

Table 85 – 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 86 – 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 110 shows the MSC of a Write service used to write the value of a single attribute, in the case of success.

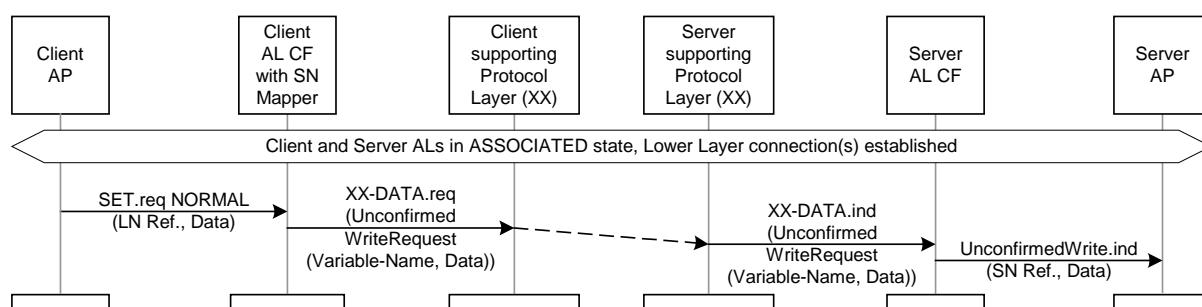


Figure 110 – MSC of the Unconfirmed Write service used for writing an attribute

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 87.

Table 87 – Mapping between the EventNotification and InformationReport services

EventNotification.ind (one or more)	InformationReport.ind
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

The general block transfer (GBT) mechanism can be used to carry any xDLMS service primitive when the service parameters are long i.e. their encoded form is longer than the Max Receive PDU Size negotiated. In this case, the AL uses one or more General-Block-Transfer (GBT) xDLMS APDUs to transport such long messages.

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:

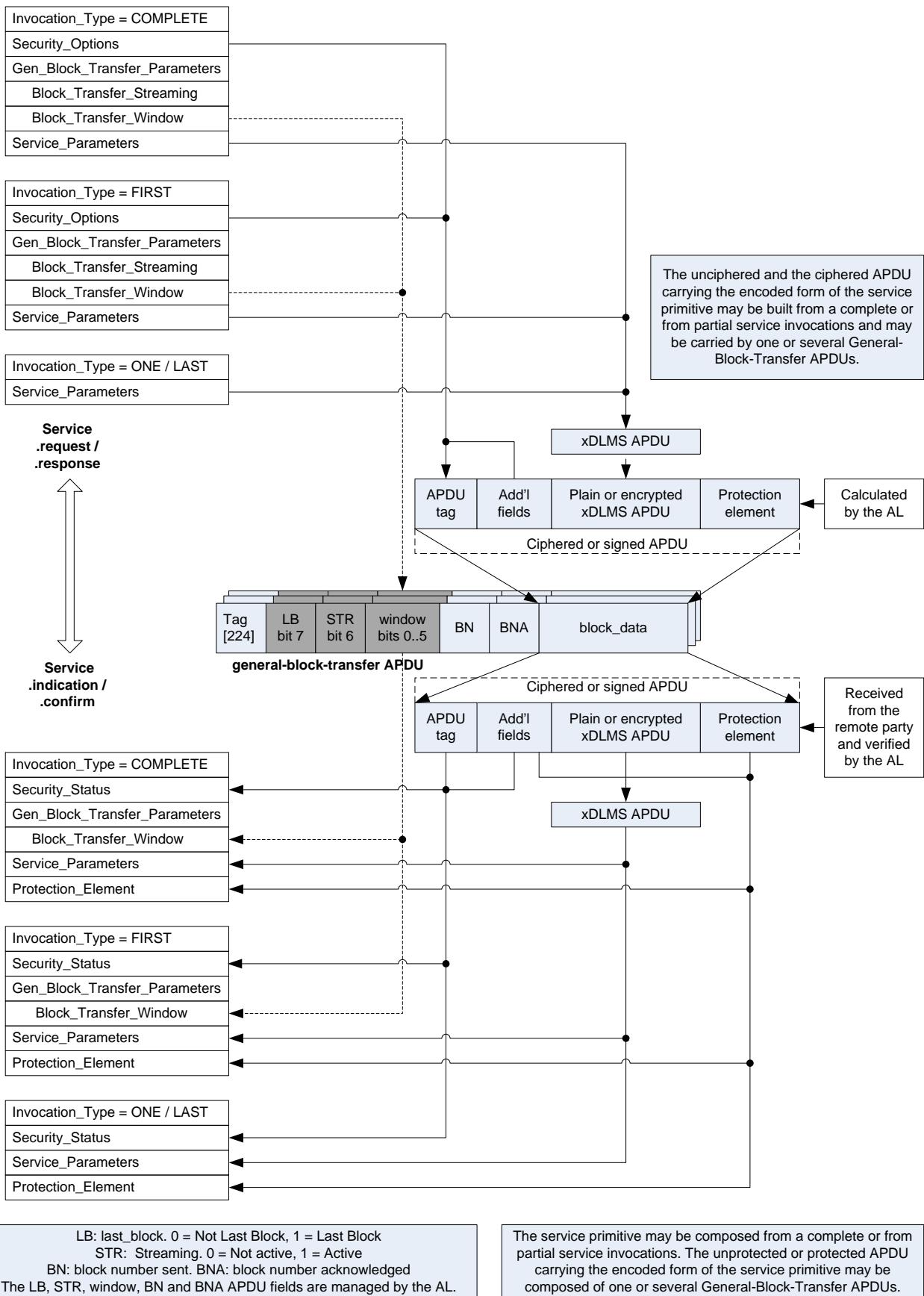
- assembles the block-data fields of the GBT APDUs received together;
- when the resulting complete APDU is ciphered, it checks and removes the protection;
- it invokes the appropriate service primitive, passing the additional Security_Status, the General_Block_Transfer_Parameters and the Protection_Element.

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 88.

A 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.

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 111 – received from the remote AL.



NOTE Applying and checking/removing cryptographic protection on APDUs is independent from the GBT process. It is included here for completeness.

Figure 111 – Partial service invocations and GBT APDUs

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 111.

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 111 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 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);
- 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. 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 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 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 increased 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.

If a party has no blocks to send, then the last-block bit of the APDU shall be set to 1 and the streaming bit shall be set to 0.

The protocol of the GBT mechanism is further explained with the help of Figure 112, Figure 113, Figure 114, Figure 115, Figure 116, Figure 117 and Figure 118. 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 2 In these examples the service-specific block transfer mechanism is not used.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	302/500
-----------------------	------------	------------------------	---------

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.

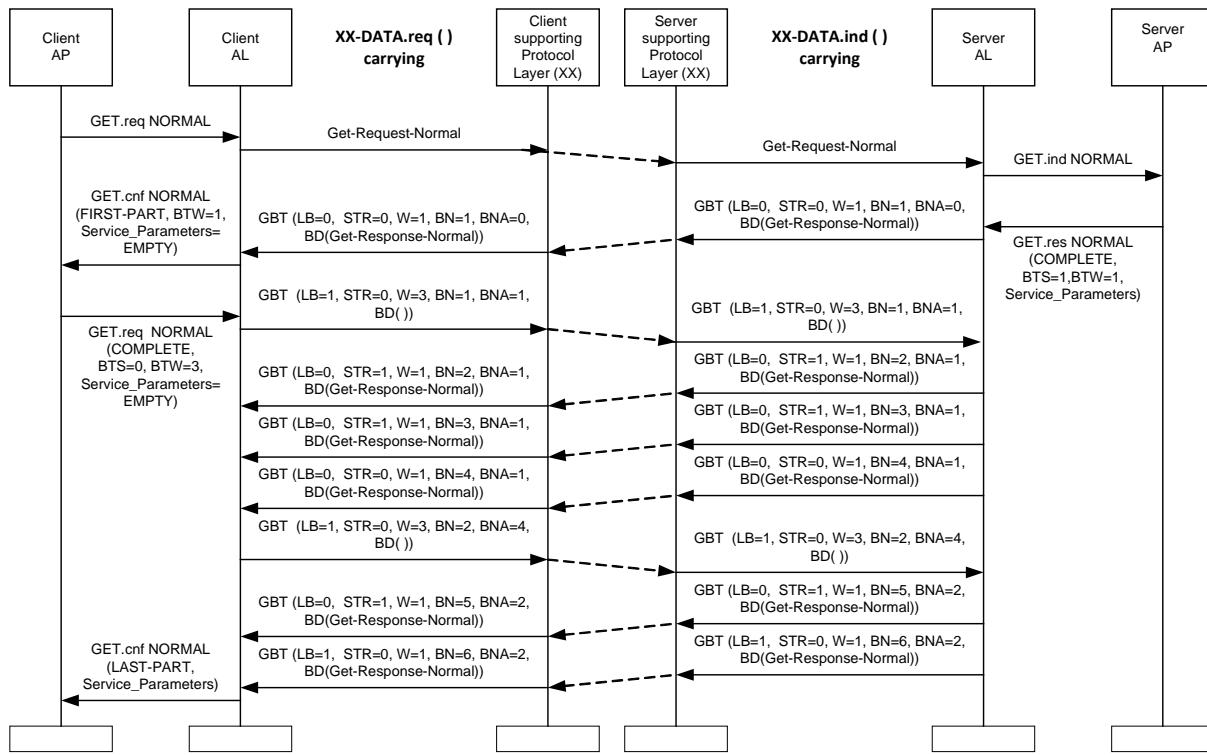
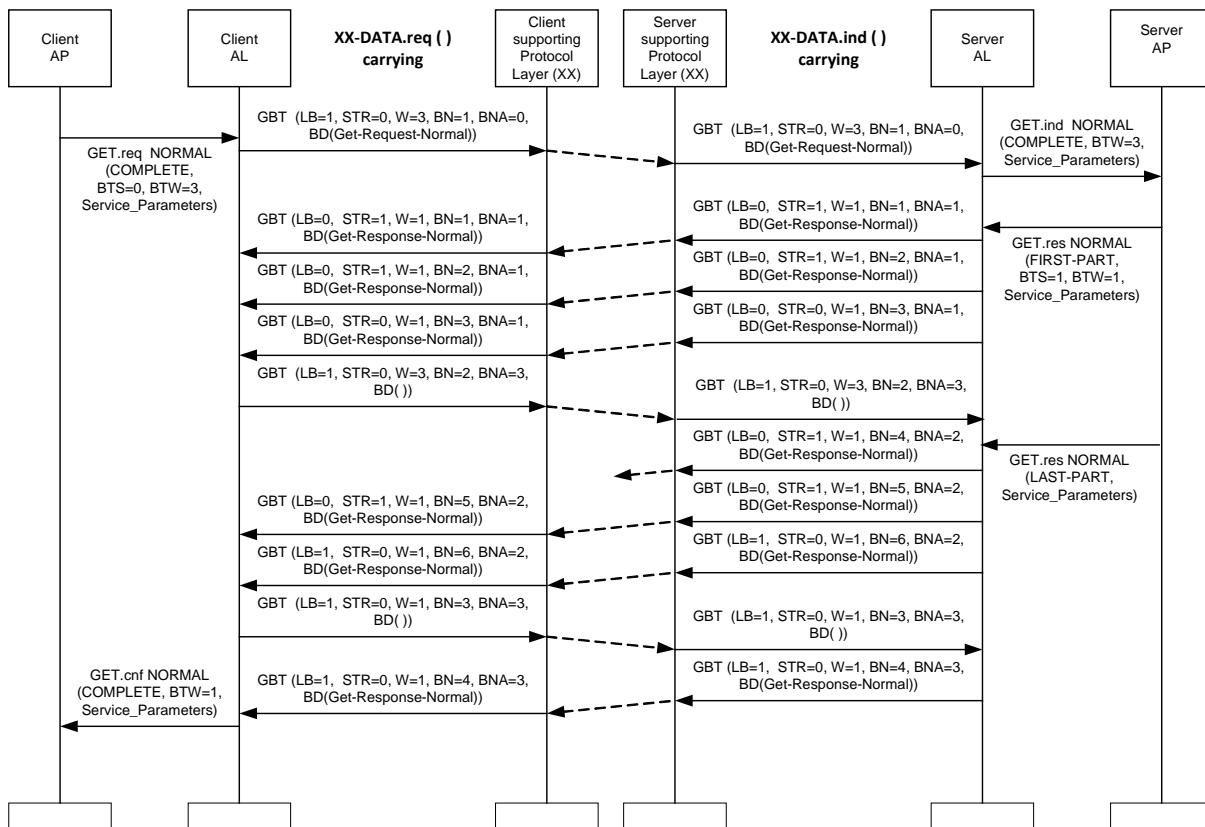


Figure 112 – GET service with GBT, switching to streaming

Figure 112 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 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 2nd (STR = 1, BN = 2), 3rd (STR = 1, BN = 3) and 4th (STR = 0, BN = 4) blocks;
 - the client AL sends a GBT APDU to confirm the reception of the 2nd, 3rd and 4th block (LB = 1, STR = 0, W = 3, BNA = 4);
 - the server AL sends the 5th (STR = 1, BN = 5) and the 6th, last block (LB = 1, STR = 0, BN = 6);
- NOTE 3 The last block is not confirmed. However, when lost, it can be recovered. See Figure 115.
- 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.



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 113 – GET service with partial invocations, GBT and streaming, recovery of 4th block sent in the 2nd stream

Figure 113 shows 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 4th 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 1st, 2nd and 3rd 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 4th, 5th and 6th block (LB = 1, STR = 0, BN = 6). However, the 4th block gets lost;

- the client AL indicates that the 4th block has not been received by sending a GBT APDU confirming the reception of the 3rd 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 4th block (LB = 1, 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.

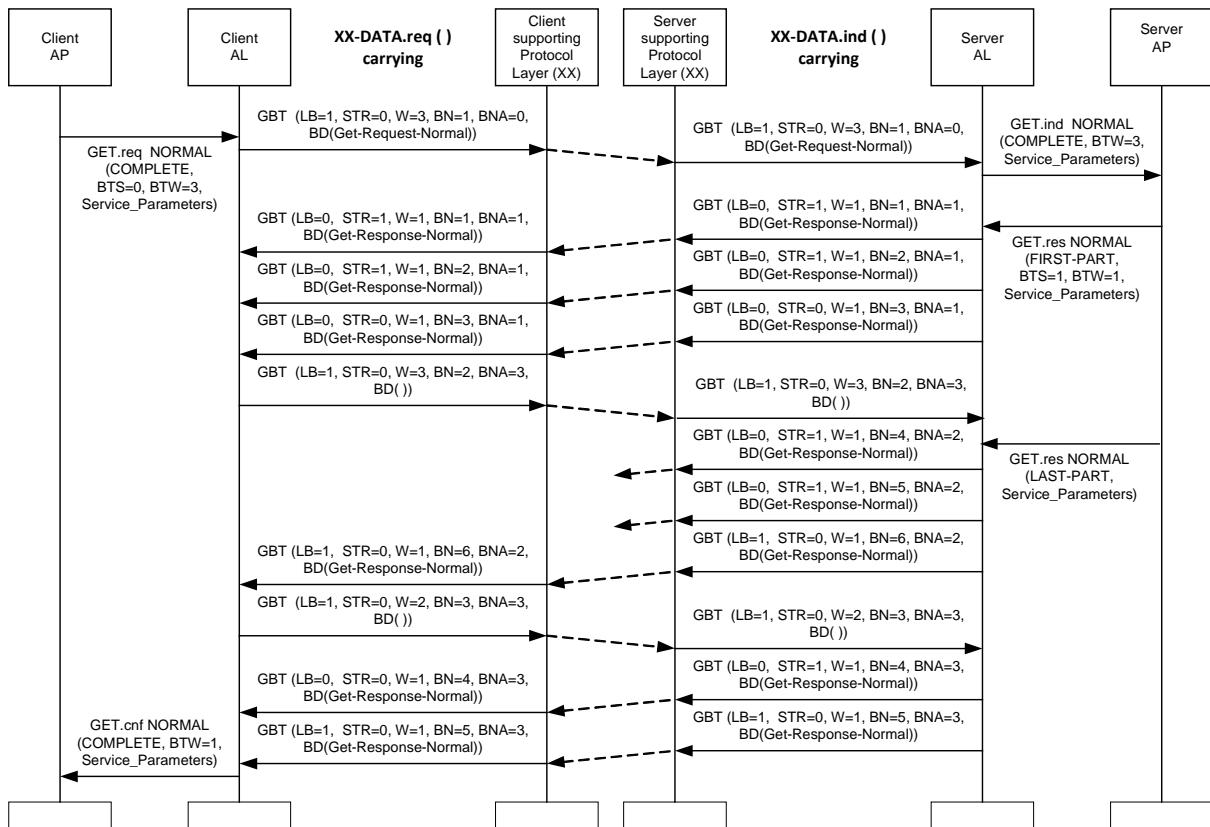
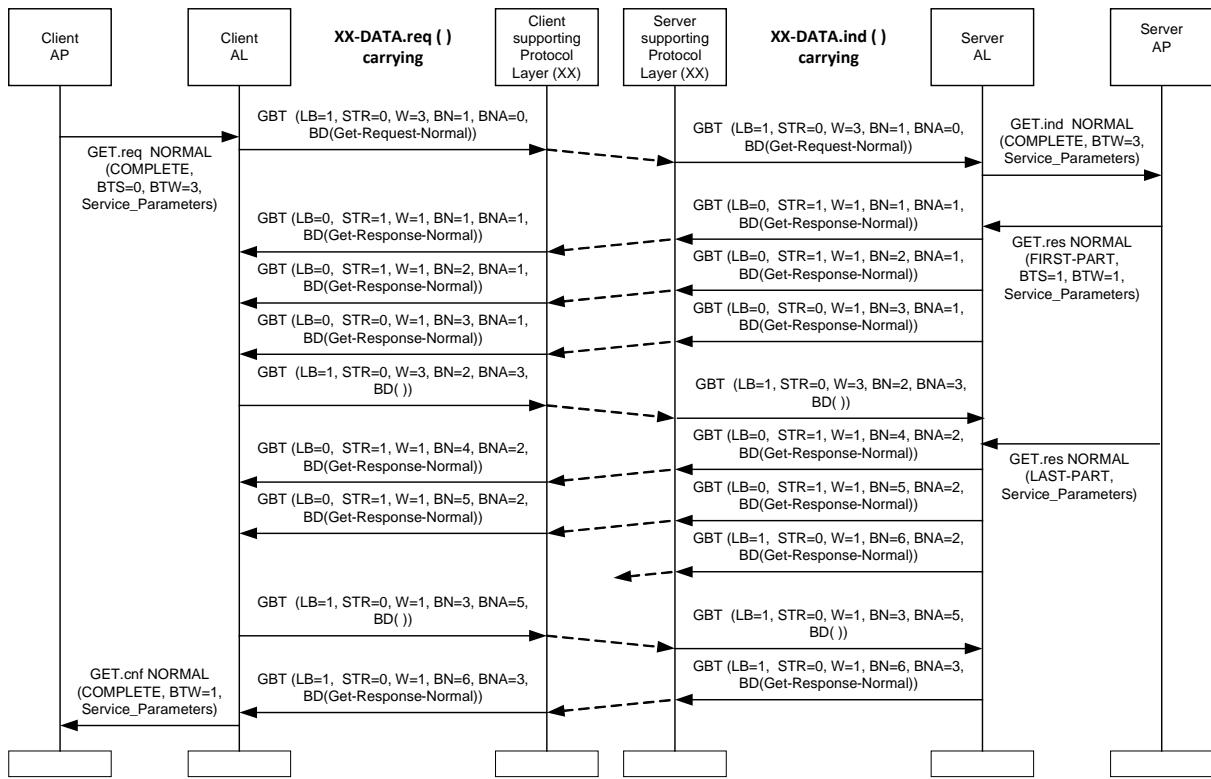


Figure 114 – GET service with partial invocations, GBT and streaming, recovery of 4th and 5th block

Figure 114 shows a scenario which is essentially the same as in Figure 113 except that the 4th and 5th blocks are lost and recovered. The process is the following:

- the client receives the 6th block (LB = 1, STR = 0, BN = 6, BNA = 2);
- the client indicates that the 4th and the 5th 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 3rd 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) 4th (LB = 0, STR = 1, BN = 4) and 5th block (LB = 1, STR = 0, BN = 5). Notice here that LB has been set to 1;
- 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.

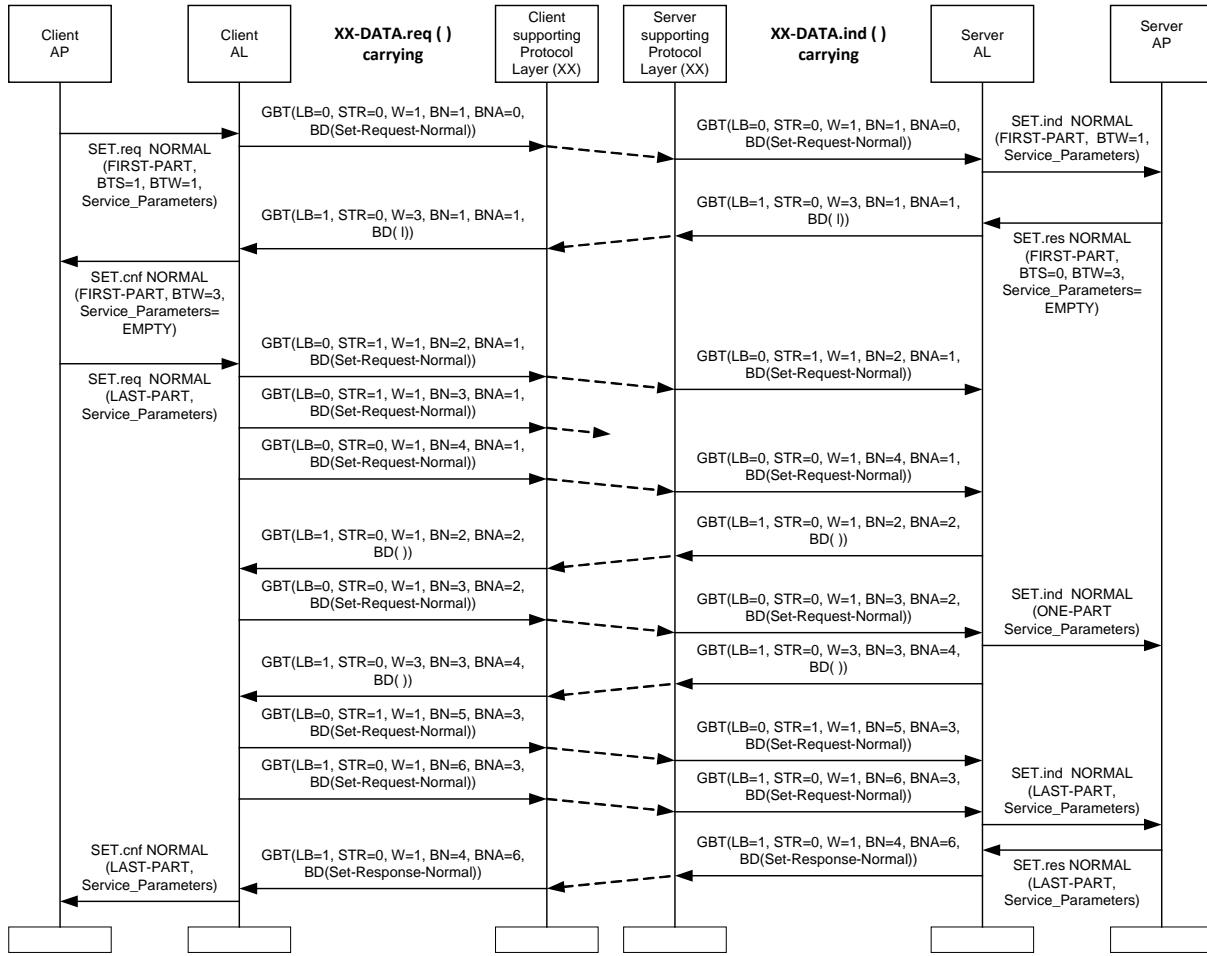


Legend in the service primitive: BTS: Block_Transfer_Streaming, BTW: Block_Transfer_Window. Legend in the APDUs: GBT: General-Block-Transfer APDU, LB: Last-Block, STR: Streaming, W: Window, BN: Block number, BNA: Block number acknowledged, BD (APDU): block-data containing one block of the APDU

Figure 115 – GET service with partial invocations, GBT and streaming, recovery of last block

Figure 115 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 5th 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) 6th 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.



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 116 – SET service with GBT, with server not supporting streaming, recovery of 3rd block

Figure 116 shows a SET service with GBT and streaming. The 3rd 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 = 1, 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 2nd, 3rd and 4th block in a stream. However, the 3rd block gets lost;
- the server indicates that block 3 is lost by confirming the reception of the 2nd block and dropping down the window size to 1 (LB = 1, 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 invokes a SET.indication NORMAL service primitive with Invocation_Type = ONE-PART. The server AL confirms the reception of the blocks up to the 4th block (LB = 1, STR = 0, W = 3, BNA = 4). Notice that the window size has been raised again to 3;

- the client sends then the 5th and the 6th block using streaming;
 - when the server AL receives the 6th, 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.request 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. The client AL invokes the SET.confirm NORMAL service primitive with Invocation_Type = LAST-PART. Service_Parameters include the result of the set

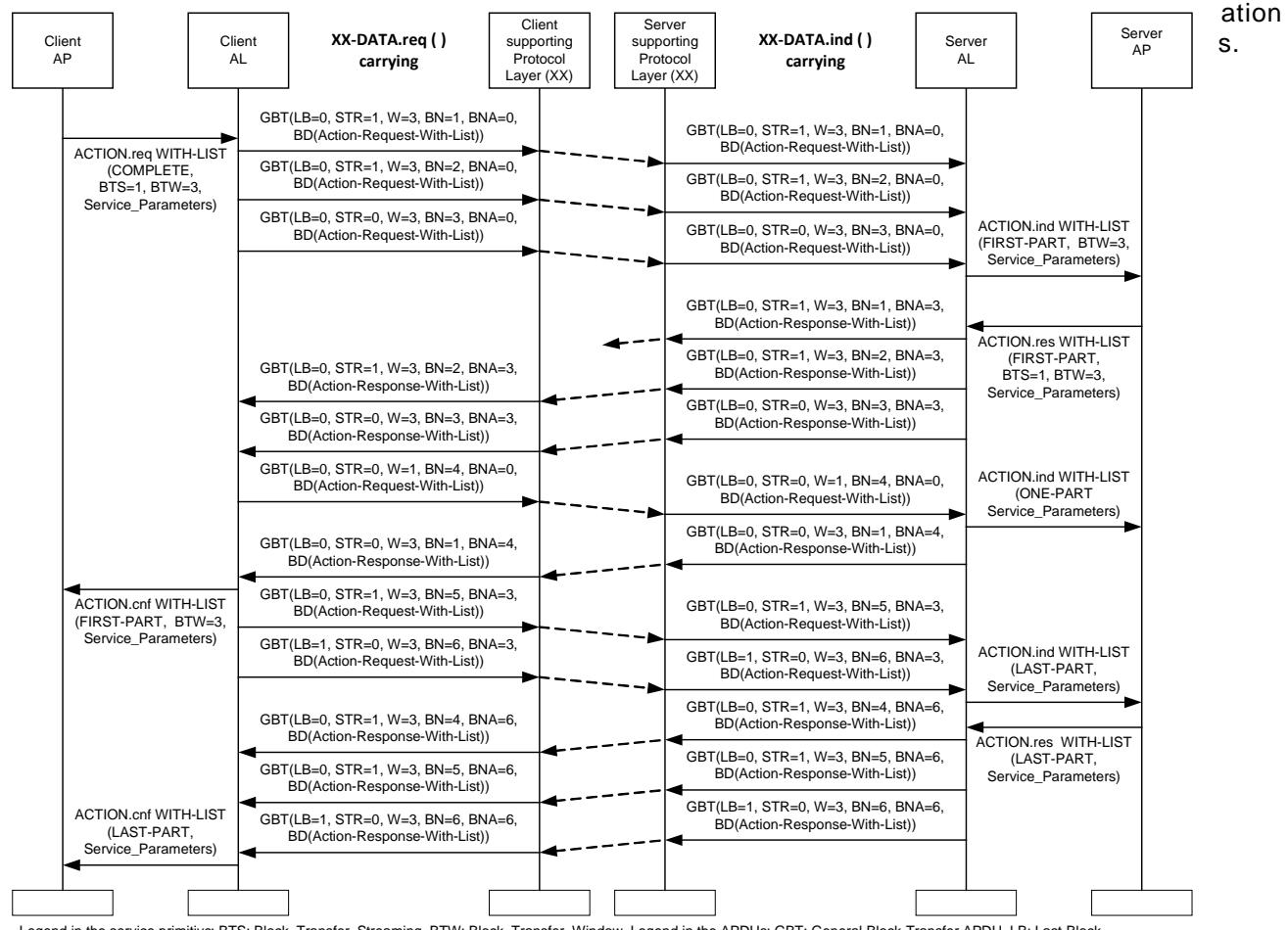


Figure 117. ACTION WITH LIST AND WITH ADDITIONAL CBT AND MARCHES

Figure 117 shows 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 1st block is lost;

- the client AL asks the server to send the lost 1st block again by not confirming any blocks received. It also sends its 4th 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 1st block and confirms the 4th 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 5th and the 6th, 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.

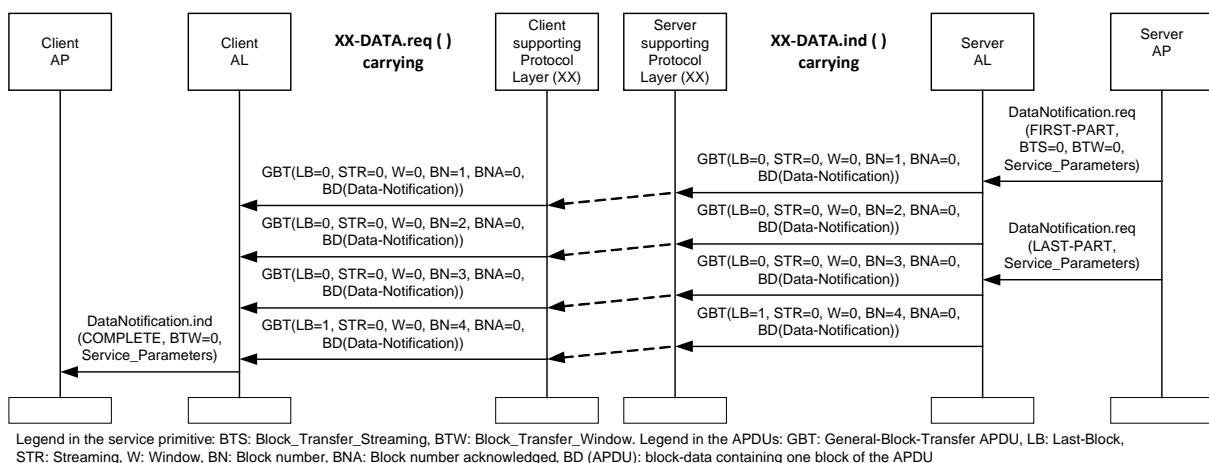


Figure 118 – DataNotification service with GBT with partial invocation

Figure 118 shows a DataNotification service with GBT, with partial service invocations on the server side. The process is the following:

- the server AP invokes a 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.

Aborting the GBT process

The client or the server may want to abort the GBT process. To do so, it shall send a GBT APDU with LB = 1, STR = 0, BN = 0 and BNA = 0. The block transfer process 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 DataNotification.

9.5 Abstract syntax of COSEM APDUs

The abstract syntax of COSEM APDUs is specified in this subclause 9.5 using ASN.1. See ISO/IEC 8824.

NOTE The CIASE APDUs are specified in 10.4.9.

```

COSEM pdu DEFINITIONS ::= BEGIN

ACSE-APDU ::= CHOICE
{
    aarq                                AARQ-apdu,
    aare                                AARE-apdu,
    rlrq                                RLRQ-apdu,
    rlrq                                RLRE-apdu
}

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,
    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

-- not used in DLMS/COSEM
    ded-initiateRequest        [65] IMPLICIT  OCTET STRING,
    ded-readRequest             [69] IMPLICIT  OCTET STRING,
    ded-writeRequest            [70] IMPLICIT  OCTET STRING,

-- not used in DLMS/COSEM
    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,
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	310/500
-----------------------	------------	------------------------	---------

```

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 ::= [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,
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	311/500
-----------------------	------------	------------------------	---------

```

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.

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),
  }
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	312/500
-----------------------	------------	------------------------	---------

```

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, 0xFA00

-- 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),
    reserved-six            (6),
    reserved-seven          (7),
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	313/500
-----------------------	------------	------------------------	---------

```

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,
getTIAtribute                  [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
  }
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	314/500
-----------------------	------------	------------------------	---------

```

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
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	315/500
-----------------------	------------	------------------------	---------

```

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,
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	316/500
-----------------------	------------	------------------------	---------

```

        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] IMPLICIT ENUMERATED
    {
        operation-not-possible    (1),
        service-not-supported     (2),
        other-reason               (3)
    }
}

-- Access

Access-Request ::= SEQUENCE
{
    long-invoke-id-and-priority    Long-Invoke-Id-And-Priority,
    date-time                      OCTET STRING,
    access-request-body            Access-Request-Body
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	318/500
-----------------------	------------	------------------------	---------

```

}

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
}

-- 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                            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
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	319/500
-----------------------	------------	------------------------	---------

```

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)),
    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),
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	320/500
-----------------------	------------	------------------------	---------

```

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),
    other-reason              (250)
}

-- IEC 61334-6:2000 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-inverse-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
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	321/500
-----------------------	------------	------------------------	---------

```

}

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
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	322/500
-----------------------	------------	------------------------	---------

```

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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	323/500
-----------------------	------------	------------------------	---------

9.6 COSEM APDU 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. In this subclause 9.6, COSEMpdu ASN.1 definition is provided and mapped to COSEMpdu XSD definition.

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">

    <!-- ASN.1 definitions -->
    <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>

    <!-- ACSE-APDU definition -->
    <xsd:element name="aCSE-APDU" type="ACSE-APDU"/>
    <xsd:complexType name="ACSE-APDU">
        <xsd:choice>
            <xsd:element name="aarq" type="AARQ-apdu"/>
            <xsd:element name="aare" type="AARE-apdu"/>
            <xsd:element name="rlrq" type="RLRQ-apdu"/>
            <xsd:element name="rlre" type="RLRE-apdu"/>
        </xsd:choice>
    </xsd:complexType>

    <!-- xDLMS-APDU definition -->
    <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="unconfirmedWriteRequest" type="UnconfirmedWriteRequest"/>
        </xsd:choice>
    </xsd:complexType>

```

```

<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: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: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"/>

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	326/500
-----------------------	------------	------------------------	---------

```

</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>

<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="reserved-six"/>
            <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:restriction>
</xsd:simpleType>

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	327/500
-----------------------	------------	------------------------	---------

```

<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: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>

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	328/500
-----------------------	------------	------------------------	---------

```

<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: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: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:choice>
  <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: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: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="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:choice>
</xsd:complexType>

```

```

<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 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:sequence>
</xsd:complexType>

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	333/500
-----------------------	------------	------------------------	---------

```

<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:restriction>
      </xsd:simpleType>
    </xsd:element>
  </xsd:choice>
</xsd:complexType>

```

```

        <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: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="getTIAtribute" type="ServiceError"/>
  <xsd:element name="changeScope" type="ServiceError"/>
  <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>

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	336/500
-----------------------	------------	------------------------	---------

```

</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>

<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:sequence>
</xsd:complexType>

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	337/500
-----------------------	------------	------------------------	---------

```

<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: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">
  <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>

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	340/500
-----------------------	------------	------------------------	---------

```

<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:sequence>

```

```

<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>

<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:simpleType>
        <xsd:restriction base="xsd:token">
          <xsd:enumeration value="operation-not-possible"/>
          <xsd:enumeration value="service-not-supported"/>
          <xsd:enumeration value="other-reason"/>
        </xsd:restriction>
      </xsd:simpleType>
    </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>

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	342/500
-----------------------	------------	------------------------	---------

```

<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>

<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>

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	343/500
-----------------------	------------	------------------------	---------

```

</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: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">

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	344/500
-----------------------	------------	------------------------	---------

```
<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: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

As explained in 3.8, the COSEM interface model for energy metering equipment, specified in DLMS UA 1000-1 has been designed for use with a variety of communication profiles for exchanging data over various communication media. As shown in 4.7, 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 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 Ed. 12.1:2015, 4.1.7, metering equipment is 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 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 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 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	346/500
-----------------------	------------	------------------------	---------

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.

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 metering 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.

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 119.

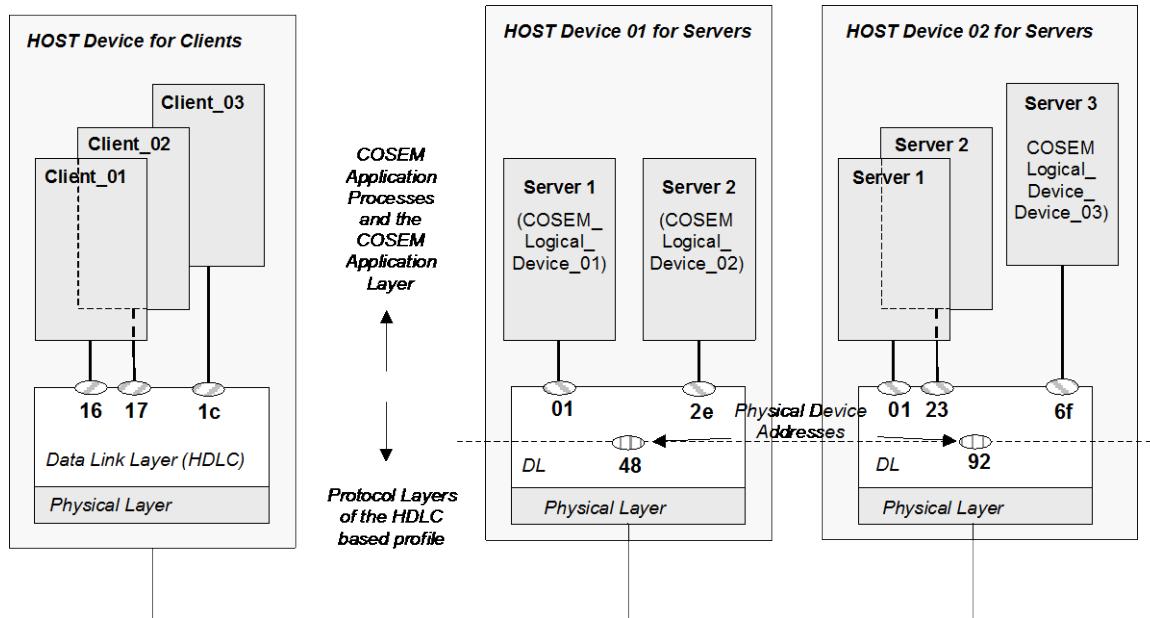


Figure 119 – Identification/addressing scheme in the 3-layer, CO, HDLC based communication profile

10.2.4 Supporting layer services and service mapping

In this profile, the supporting layer of the DLMS/COSEM AL is the HDLC based data link layer. It provides services for:

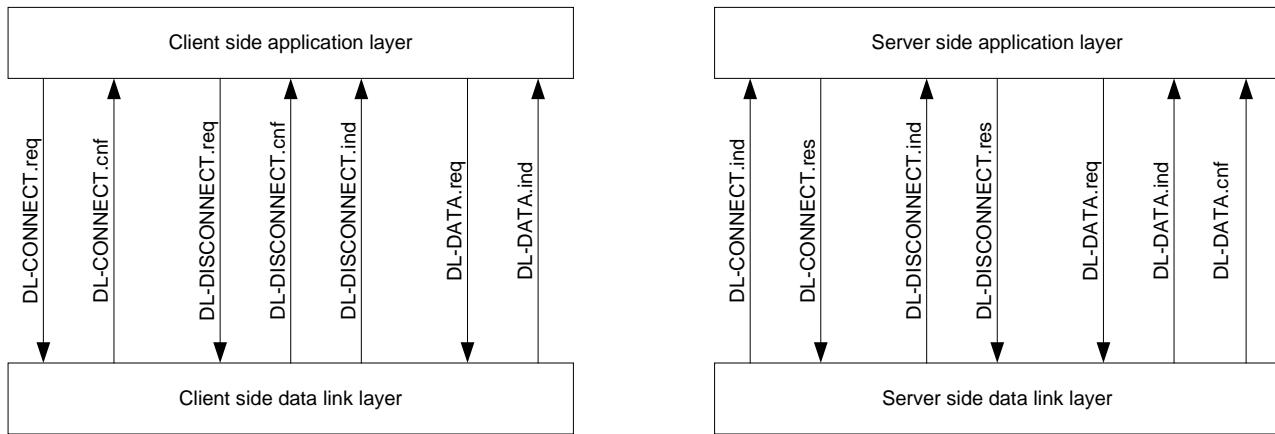
- data link layer connection management;
- connection-oriented data transfer;
- connection-less data transfer.

Figure 120 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 120 – 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 88 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 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 meter is accessed via a gateway – see 10.7 – 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 between the gateway and the meter the 3-layer, C.O. HDLC based profile is used, the gateway always places the APDU received to an I frame and forwards it to the meter.

When the meter receives an AARQ APDU carried by an I frame it shall check the response-allowed field of the xDLMS InitiateRequest APDU. If it is set to FALSE, it shall not respond.

In the case when the meter receives an xDLMS APDU in an I frame it shall check the service-class bit of the Invoke-Id-And-Priority / Long-Invoke-Id-And-Priority field when this field is present. If it is set to 0, it shall not respond.

When the meter receives an AARQ or an xDLMS APDU in a UI frame, it shall not respond.

Table 88 – 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	"UI" frame
	Unconfirmed	Send AARQ in a "UI" frame	Unconfirmed	Confirmed (not allowed)	-
				Unconfirmed	"UI" frame

NOTE As described above this table, when the meter 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.

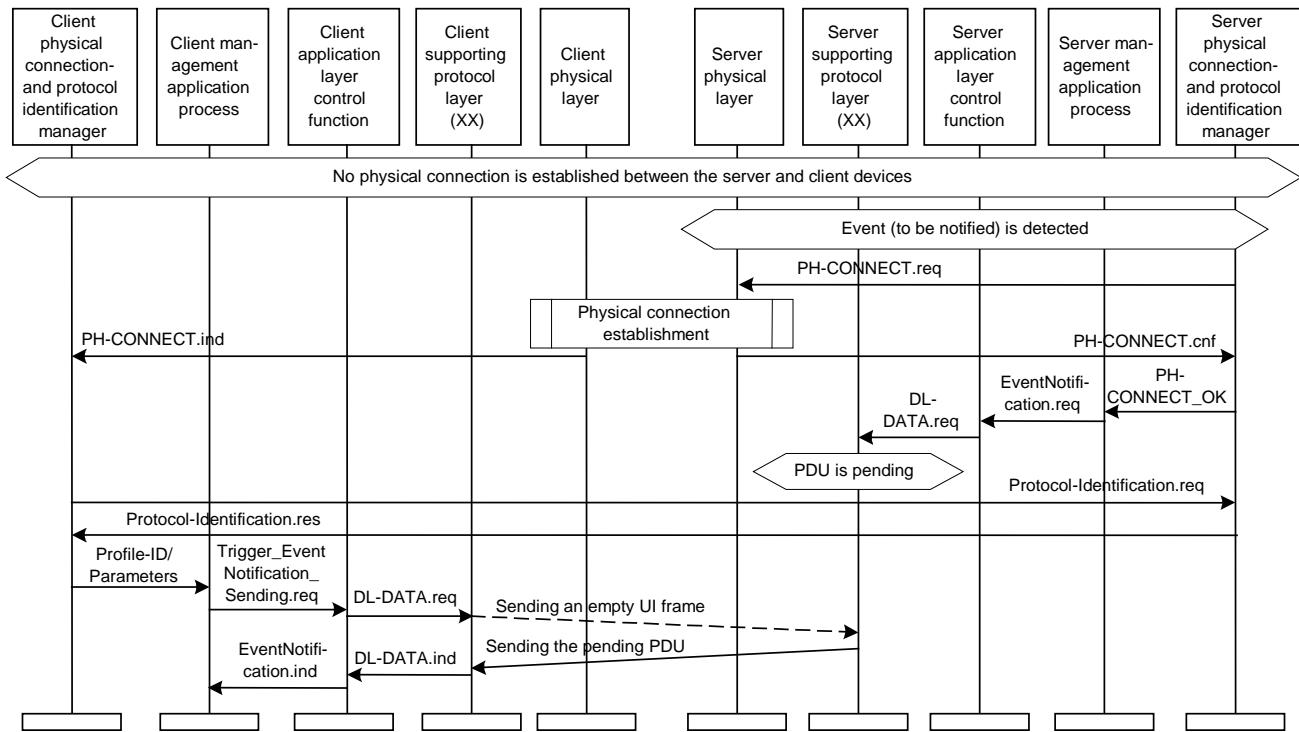
In this profile, an event is reported always by the server Management Logical Device (mandatory, reserved upper HDLC address 0x01) to the Client Management AP (mandatory, reserved HDLC address 0x01).

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.

Figure 121 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 121 – 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 121. 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 physical metering equipment, using a shared communication resource like a telephone modem. Various physical arrangements are available, like a star, daisy chain or a bus topology. These arrangements can be modelled with a logical bus, to which the metering equipment and the shared resource are connected, see Figure 122.

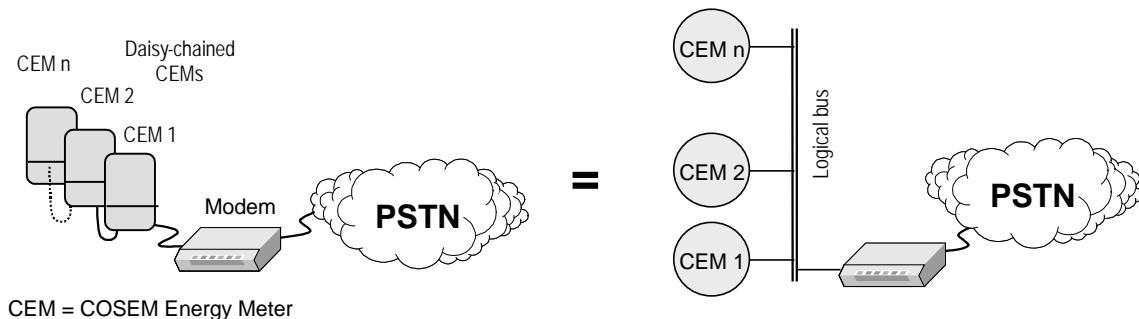


Figure 122 – Multi-drop configuration and its model

As collision on the bus must be avoided, but a protocol controlling access to the shared resource is not available, access to the bus must be controlled by external rules. In most cases, a Master-Slave arrangement is used, where the metering equipment 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. Obviously, 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.

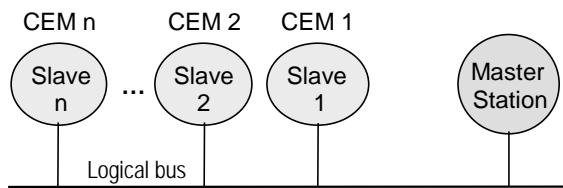


Figure 123 – Master/ Slave operation on the multi-drop bus

The client may send requests at the same time to multiple servers, if no response is expected (multi-cast or broadcast). If the client expects a response, it must send the request to a single server, giving also the right to talk. It has to wait then for the response before it may send a request to another server and with this, giving the right to talk. Arbitration of access to the common bus is thus controlled in a time-multiplexing fashion.

Messages from the client to the servers must contain addressing information. In this profile, it is ensured by using HDLC addresses. If a multi-drop arrangement is used, the HDLC address is split to two parts: the lower HDLC address to address physical devices and the upper HDLC address to address logical devices within the physical device. Both the lower and the upper address may contain a broadcast address. For details, see 8.4.2.

To be able reporting events, a server may initiate a connection to the client, using the unsolicited EventNotification / InformationReport services. As events in several or all meters connected to a multidrop may occur simultaneously – for example in the case of a power failure – they may initiate a call to the client simultaneously. For such cases, two 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 (for example sending AT commands to the modem) simultaneously. 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.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	354/500
-----------------------	------------	------------------------	---------

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 metering equipment via IP enabled networks such as Wide Area Networks, Neighbourhood Networks or Local Networks. This is shown in Figure 124.

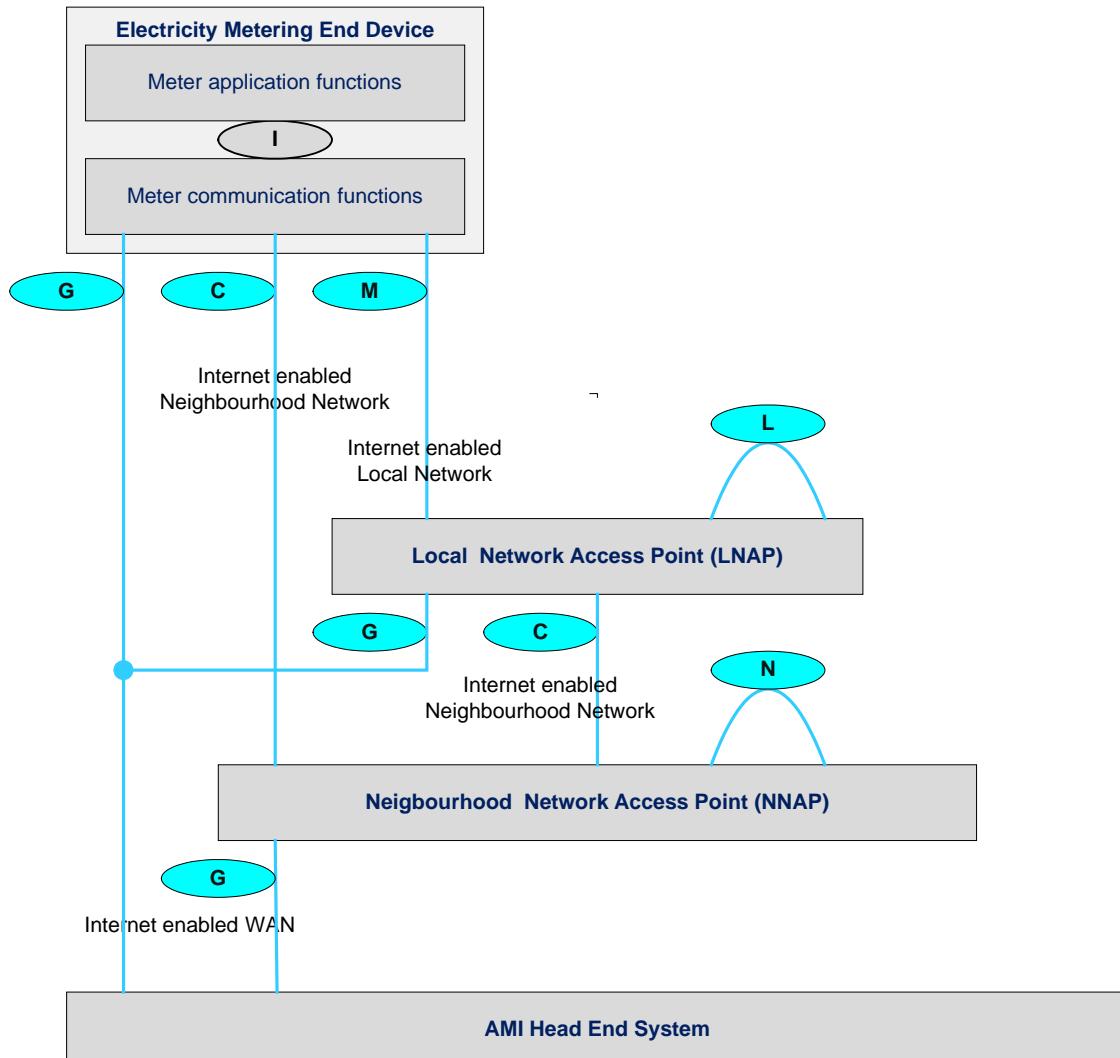


Figure 124 – Communication architecture

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.

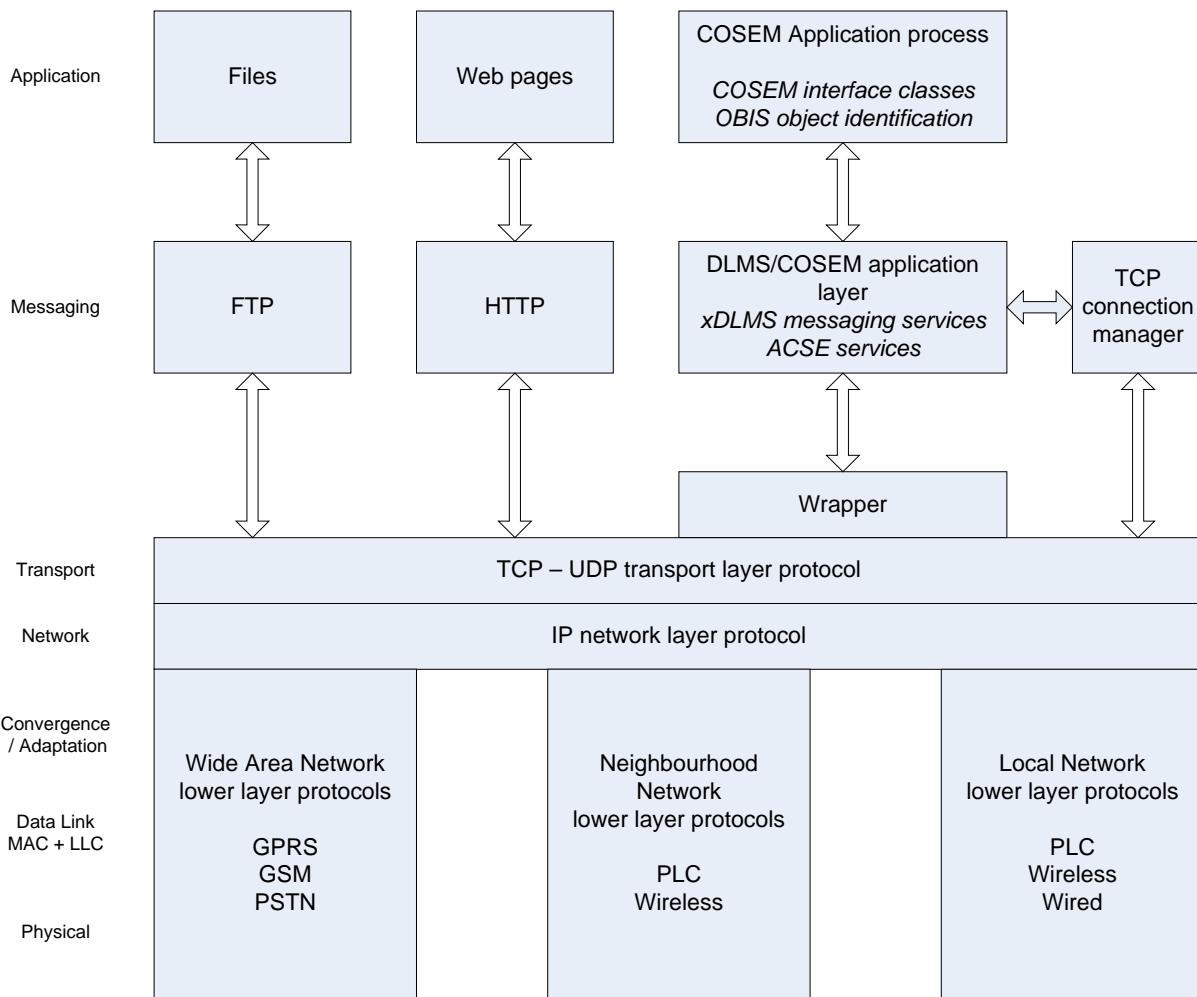


Figure 125 – Examples for lower-layer protocols in the TCP-UDP/IP based profile(s)

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 125 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 126 shows an example.

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:

```
AA 1:      { ( 163.187.45.19, T_N, 31 ) ( 163.187.45.36, T_M, 527 ) }
AA 2:      { ( 163.187.45.19, T_N, 31 ) ( 163.187.45.78, T_M, 3013 ) }
```

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.

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.

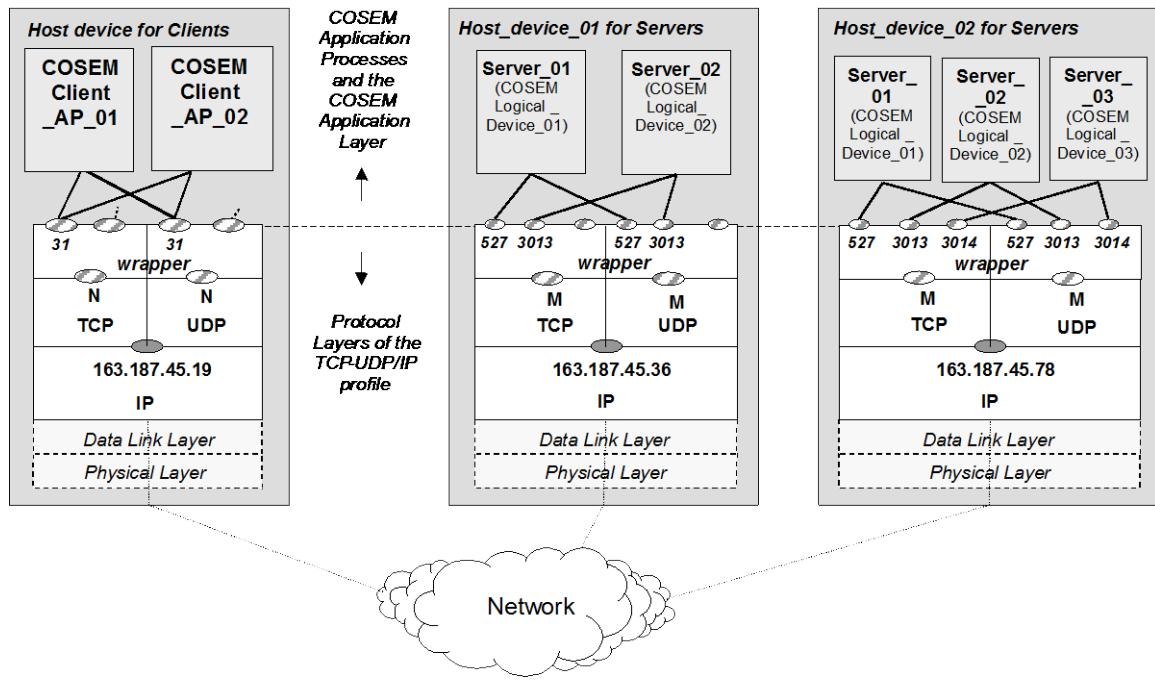


Figure 126 – Identification / addressing scheme in the TCP-UDP/IP based profile(s)

10.3.4 Supporting 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 127 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.

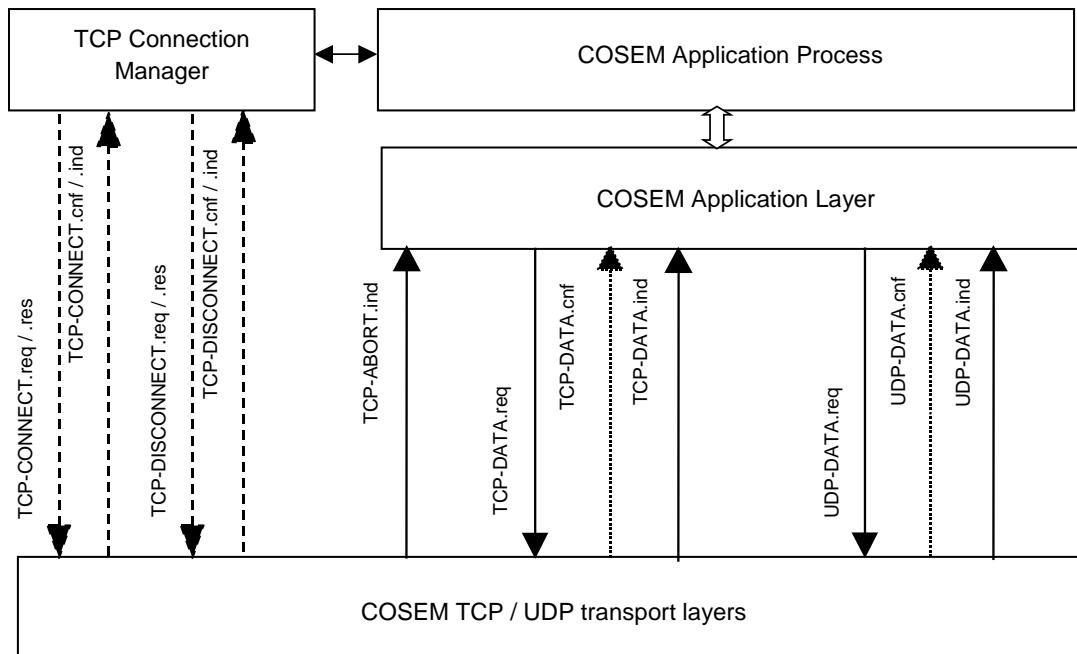


Figure 127 – Summary of TCP / UDP layer services

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 89 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 89 – 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.

As in this profile both the TCP and UDP profile allows sending data in an unsolicited manner, the TriggerEventNotificationSending service is not used.

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 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 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

Clause 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.

10.4 The S-FSK PLC profile

10.4.1 Terms and definitions relevant for the S-FSK PLC profile

10.4.1.1 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.

10.4.2 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)
SMAE	Systems Management Application Entity
SMAP	Systems Management Application Process

10.4.3 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 128.

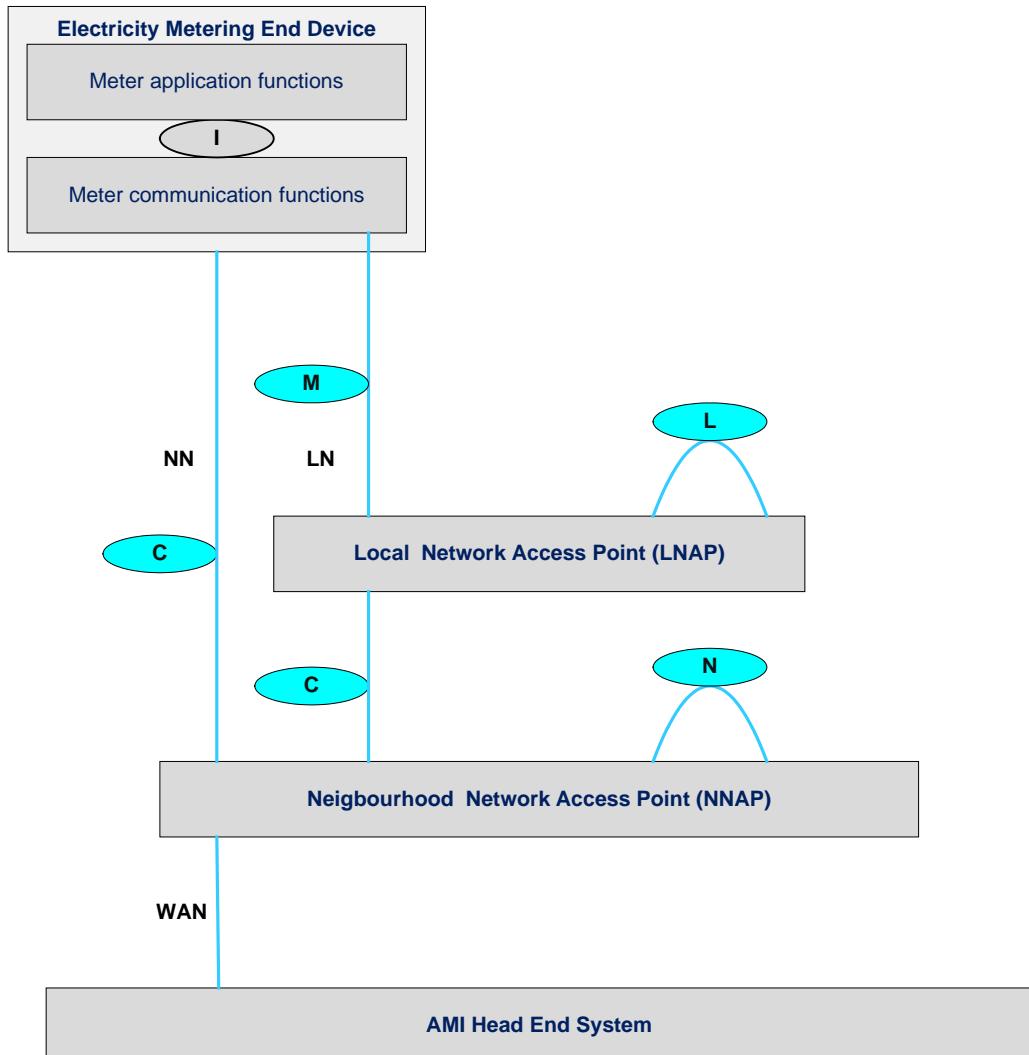


Figure 128 – Communication architecture

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/COSEM client. The end devices or the LNAPs act as DLMS/COSEM servers.

10.4.4 Reference model

NOTE This subclause 10.4.4 is partly based on IEC 61334-4-1:1996 Clause 3.

10.4.4.1 Introduction

The reference model of the *DLMS/COSEM S-FSK PLC communication profile* is shown in Figure 129. 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.

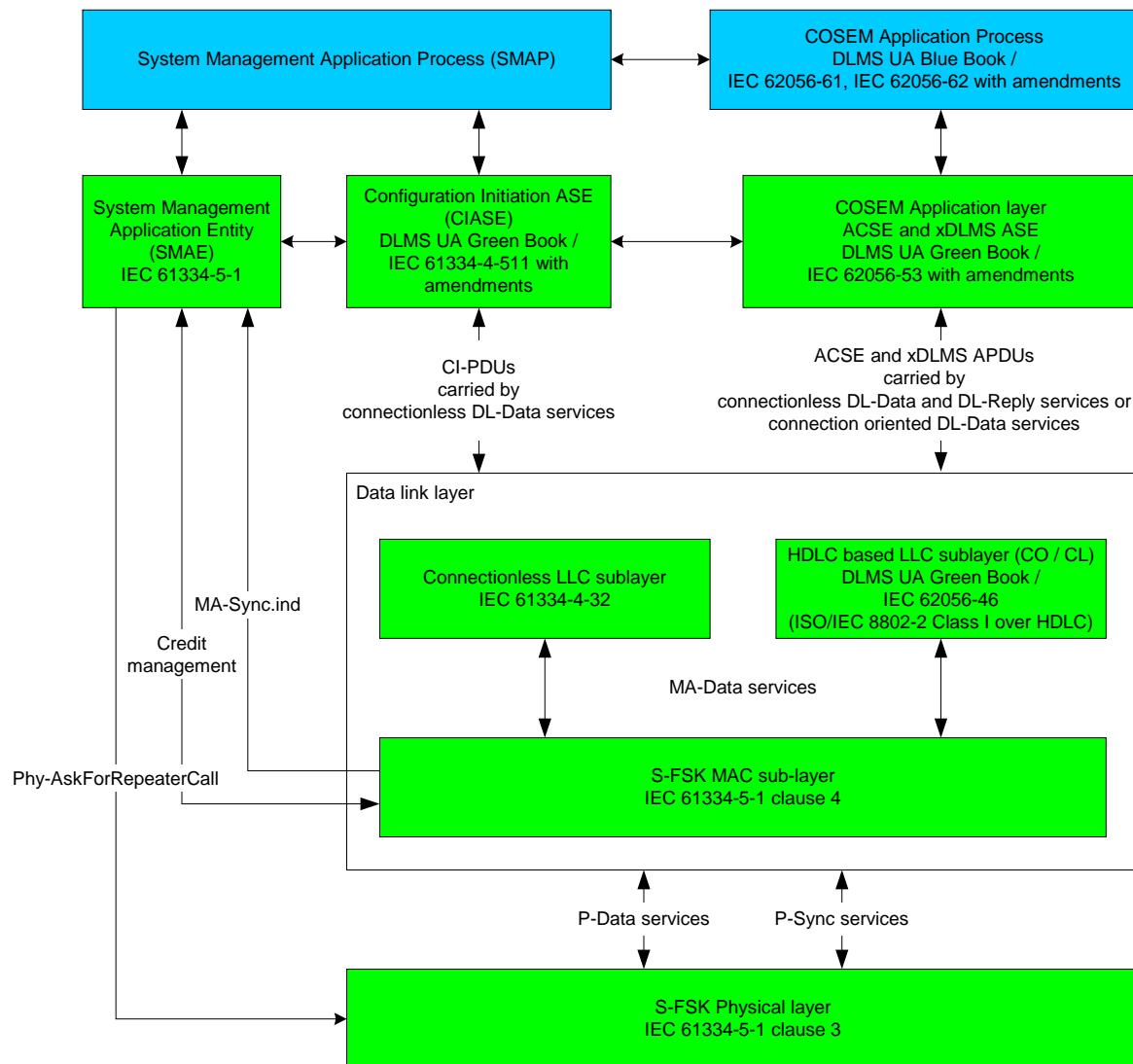


Figure 129 – The DLMS/COSEM S-FSK PLC communication profile

10.4.4.2 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.

10.4.4.3 The data link layer

10.4.4.3.1 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.

10.4.4.3.2 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.

10.4.4.3.3 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.

10.4.4.3.4 The HDLC based LLC sublayer

The HDLC based LLC sublayer is as specified in 8.

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.

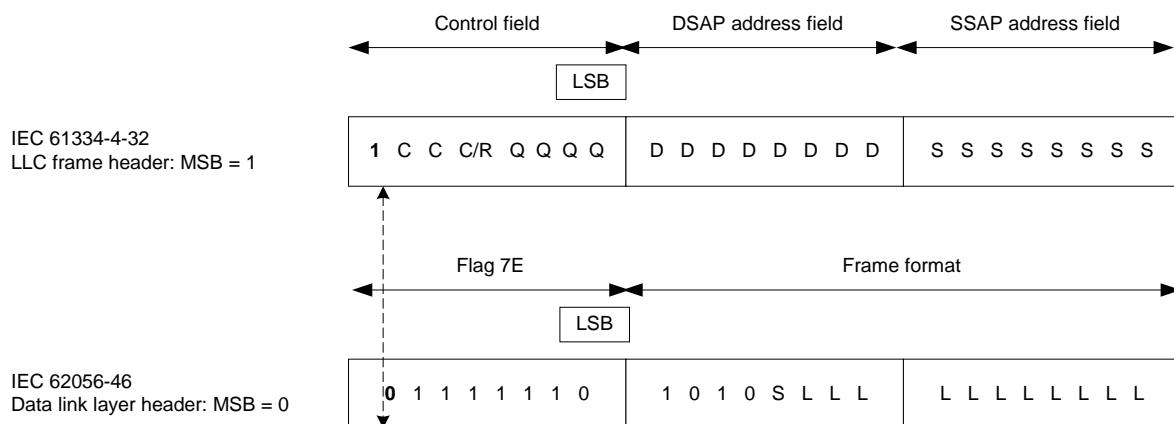
NOTE In this profile, there are two MAC sublayers. The HDLC MAC sublayer provides reliable LLC data transport and segmentation. The Medium Access Control functionality is provided by the S-FSK MAC sublayer specified in 10.4.4.3.2.

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.

10.4.4.3.5 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 130. This allows systems using the two profiles to co-exist on the same network.



Legend:

C: Command subfield
C/R: Command / response bit
QQQQ: Qualifier subfield
DDDDDDDD: Destination address
SSSSSSSS: Source address
S: Segmentation
L: Length

Figure 130 – Co-existence of the connectionless and the HDLC based LLC sublayers

10.4.4.4 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.

10.4.4.5 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.

10.4.5 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.

10.4.5.1 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 10.4.5.7, constitute upper compatible functional extensions to IEC 61334-4-511:2000.

The CIASE uses the connectionless DL-Data services of the LLC sublayer.

10.4.5.2 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 90.

Table 90 – 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.				

10.4.5.3 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 91.

Table 91 – 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.

10.4.5.4 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

The PING service primitives shall provide parameters as shown in Table 92.

Table 92 – 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.

10.4.5.5 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

The RepeaterCall service primitives shall provide parameters as shown in Table 93.

Table 93 – Service parameters of the RepeaterCall service primitives

	. request	. indication
Arguments		
Max_Adress_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_Adr_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:

$$Nb_Tslot = \lfloor MaxAdrMax / 21 \rfloor + 1$$

where $\lfloor x \rfloor$ means the floor of x, the nearest integer $\leq x$.

Example 1:

Max_Adr_MAC = 20 (20 servers on the network)

$$Nb_Tslot = \lfloor 20 / 21 \rfloor + 1 = 1$$

Nb_Tslot = 1, with 1 sub-timeslot for the concentrator and 20 sub-timeslots for the servers.

Example 2:

Max_Adr_MAC = 21 (21 servers on the network)

$$Nb_Tslot = \lfloor 21 / 21 \rfloor + 1 = 2$$

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 μ 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:

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	370/500
-----------------------	------------	------------------------	---------

- 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 μ 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.

10.4.5.6 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

The ClearAlarm service primitives shall provide parameters as shown in Table 94.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	371/500
-----------------------	------------	------------------------	---------

Table 94 – 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.

10.4.5.7 The Intelligent Search Initiator process

10.4.5.7.1 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.

10.4.5.7.2 Operation

10.4.5.7.2.1 Flow chart

The intelligent search initiator process comprises two phases:

- the *Search Initiator* phase;
- the *Check Initiator* phase.

The Intelligent Search Initiator process is shown in Figure 131. See also Figure 132 showing the complete discovery and registration process.

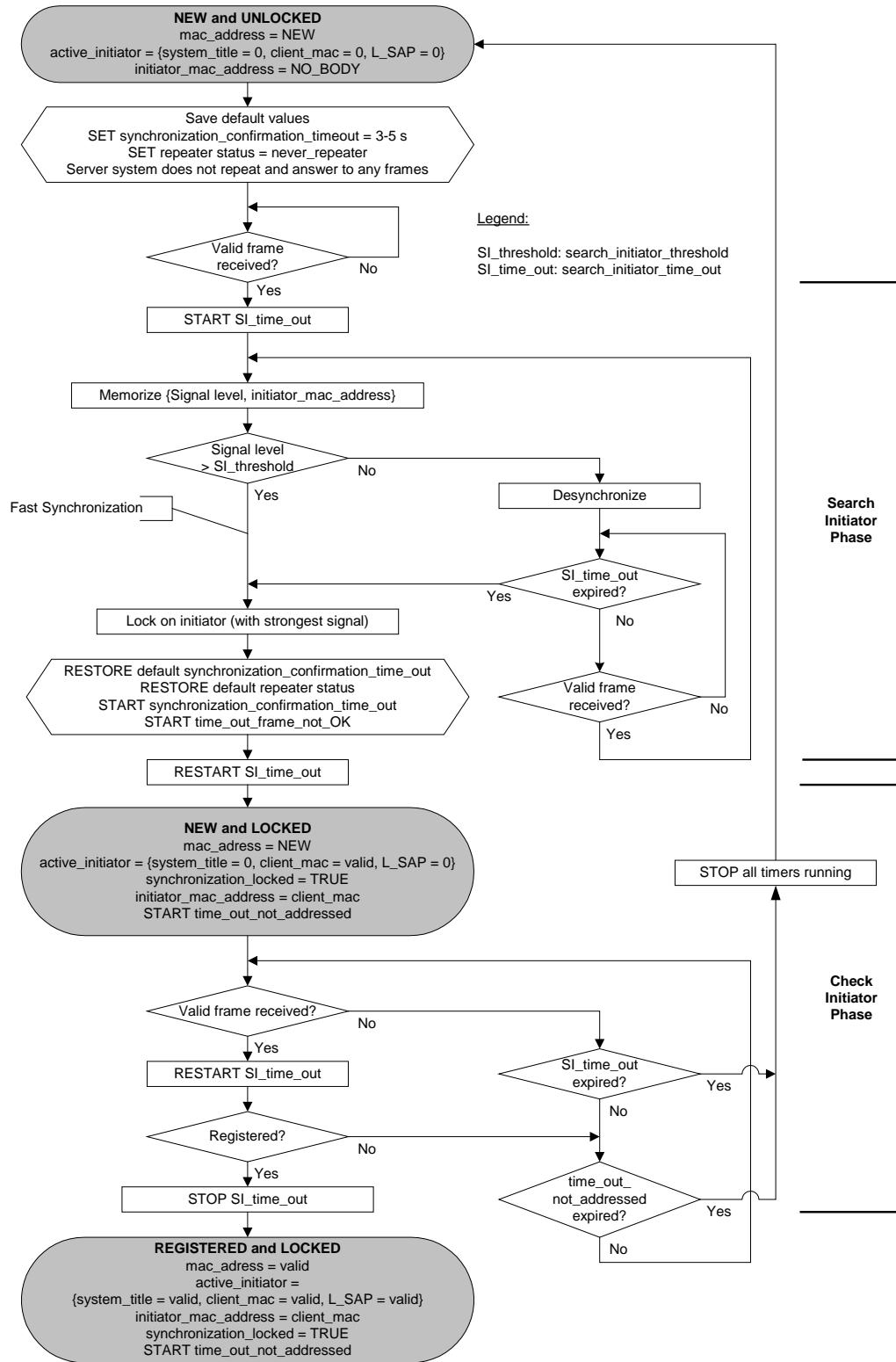
10.4.5.7.2.2 Process parameters

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 10.4.5.7.2.4;
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 μ V.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	373/500
-----------------------	------------	------------------------	---------



NOTE A valid frame is a frame in which either the source or the destination address is an initiator address, and otherwise correct.

Figure 131 – Intelligent Search Initiator process flow chart

10.4.5.7.2.3 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 ($\text{signal level} > \text{search_initiator_threshold}$ – see 10.4.5.7.2.2) 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 ($\text{signal level} < \text{search_initiator_threshold}$), the server system memorizes the signal level and the MAC address, then desynchronizes immediately in order to listen to another frame;
- when the $\text{search_initiator_time_out}$ expires, the server system locks to the initiator having provided the best signal level.

At this point:

- the default values of the $\text{synchronization_confirmation_time_out}$ and the repeater status are restored;
- the $\text{synchronization_confirmation_time_out}$ and the $\text{time_out_frame_not_OK}$ timers are initialised;
- the $\text{search_initiator_time_out}$ is restarted.

The server is in the NEW and LOCKED state and enters the Check Initiator Phase.

10.4.5.7.2.4 Check Initiator Phase

Once the server system is locked, it can be discovered and registered. The process is the following:

- the $\text{time_out_not_addressed}$ timer is started;
- the server waits then to be discovered and registered;
- the $\text{search_initiator_time_out}$ 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 $\text{search_initiator_time_out}$ timer is stopped;

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	375/500
-----------------------	------------	------------------------	---------

- 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.

10.4.5.7.2.5 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.

10.4.5.8 The Discovery and Registration process,

The Discovery and Registration process, including the Intelligent Search Initiator process, is summarized in Figure 132.

NOTE 1 The state transitions caused by writing the mac-address and initiator-mac-address variables are not shown.

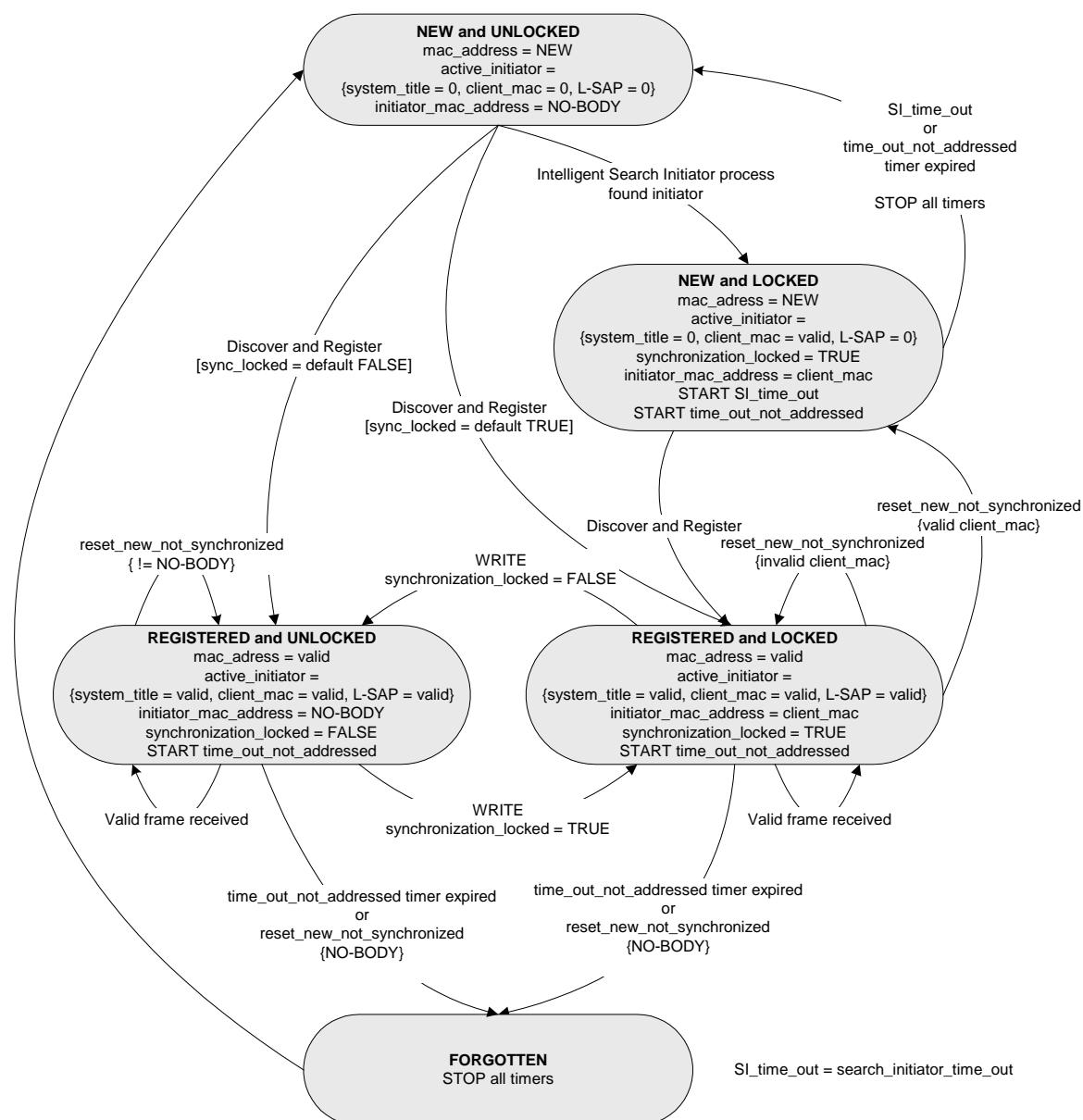


Figure 132 – The Discovery and Registration process

The process is the following.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	376/500
-----------------------	------------	------------------------	---------

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 131. 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);

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	377/500
-----------------------	------------	------------------------	---------

- 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.

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;

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	378/500
-----------------------	------------	------------------------	---------

- 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.

10.4.5.9 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 10.4.9.

10.4.6 Addressing

10.4.6.1 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.

10.4.6.2 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 95 – 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 + ... 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		

10.4.6.3 Reserved special LLC addresses

NOTE This subclause 10.4.6.3 is based on IEC 61334-4-1:1996. 4.4.

10.4.6.3.1 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).

10.4.6.3.2 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 96 and Table 97 respectively.

Table 96 – 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 97 – 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)

10.4.6.3.3 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 98 and Table 99 respectively.

Table 98 – 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 99 – 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)

10.4.6.4 Source and destination APs and addresses of CI-PDUs

The Source and destination APs and addresses of CI-PDU requests are show in Table 100.

Table 100 – 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 ¹
discoverReport	Manager	AIISMAE	NEW FFE ² or individual server MAC	FFF ³ All-Physical or Initiator	0xFD	0x00 CIASE
register	Initiator	AIISMAE	Initiator	FFF All-Physical	0x00 CIASE	0x01 CIASE ⁴
pingRequest	Initiator	Individual	Initiator	Individual	0x00 CIASE	0x01 CIASE ⁴

CI-PDU	Source AP	Destination AP	MAC SA	MAC DA	D-L-SAP	S-L-SAP
pingResponse	Individual	Initiator	Individual	Initiator	0x01 CIASE ⁴	0x00 CIASE
repeaterCall	Initiator	AIISMAE	Initiator	FFF All-Physical	0x00 CIASE	0x01 CIASE ⁴
clearAlarm	Initiator	AIISMAE	Initiator	FFF All-Physical	0x00 CIASE	0x01 CIASE ⁴

¹⁾ Could be a different value
²⁾ FFE if the server is in the NEW state. Individual MAC address if the server is in ALARM state.
³⁾ If the reporting system list feature is used, then the Destination Address is All-Physical. Otherwise, it is the Initiator address.
⁴⁾ Could be different value, but must be the same as in the discover CI-PDU.

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.

10.4.7 Specific considerations / constraints for the IEC 61334-4-32 LLC sublayer based profile

10.4.7.1 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 133.

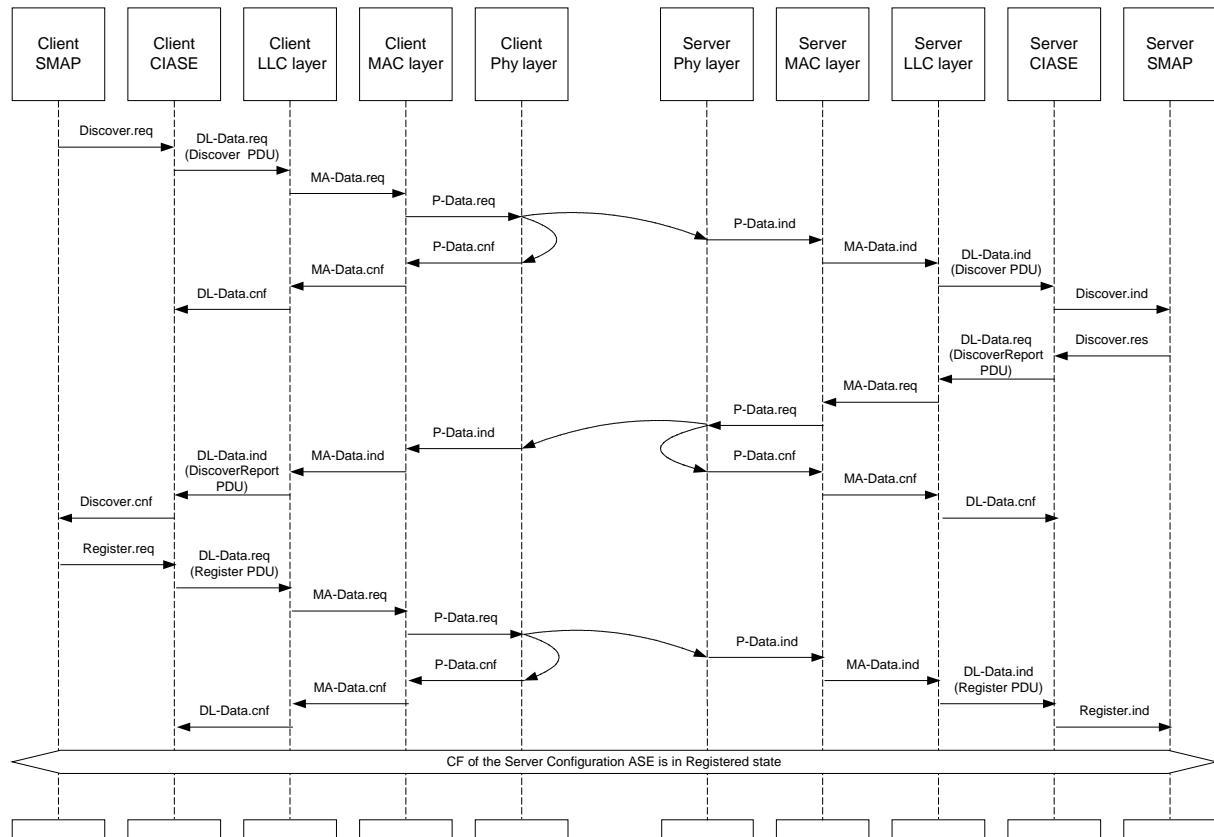


Figure 133 – MSC for the discovery and registration process

The MSC for the establishment of a confirmed AA establishment is shown in Figure 134.

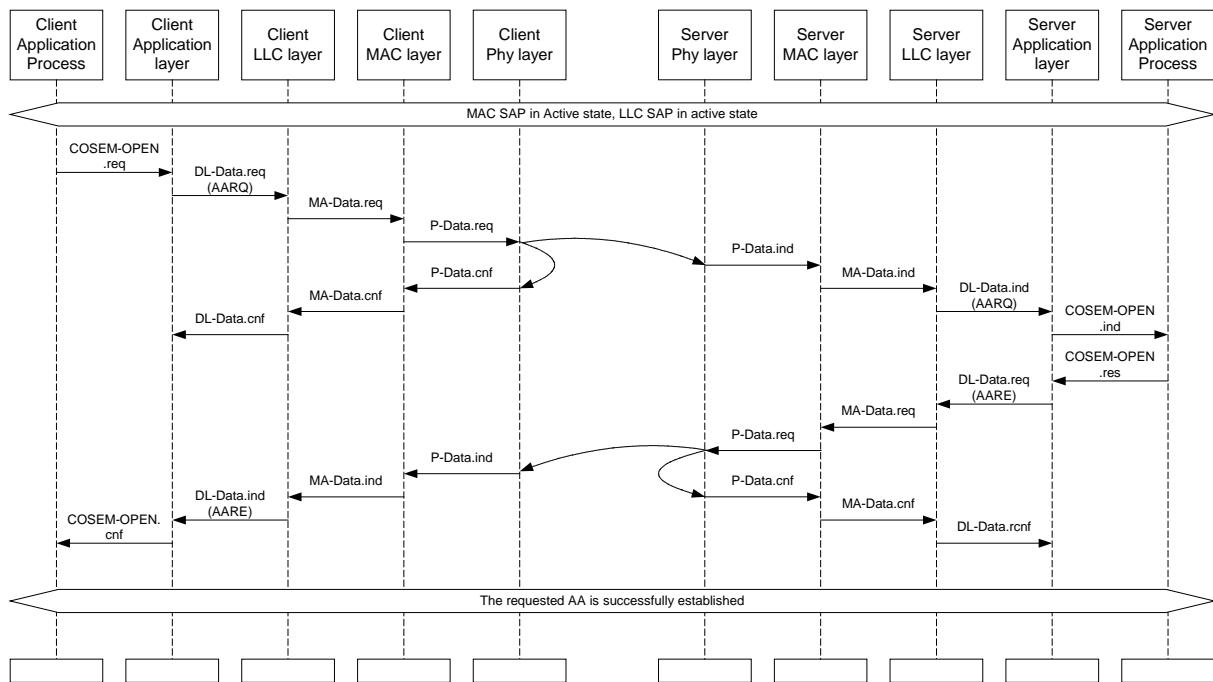


Figure 134 – MSC for successful confirmed AA establishment

10.4.7.2 Application association types, confirmed and unconfirmed xDLMS services

Table 89 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. 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.

Table 101 – 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

10.4.7.3 xDLMS request/response type services

No specific features / constraints apply related to the use of request/response type services.

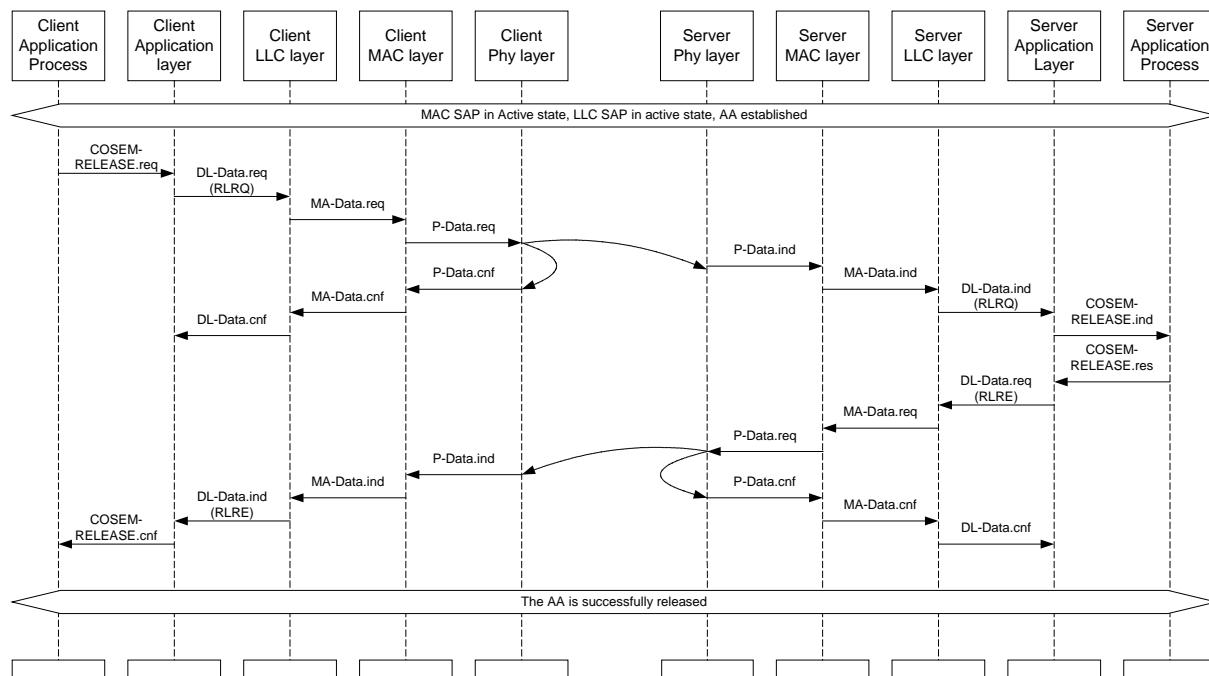
The MSC is essentially the same as for establishing confirmed AAs, except that instead of the COSEM-OPEN service primitives, the appropriate xDLMS service primitives are used.

10.4.7.4 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 135.

**Figure 135 – MSC for releasing an Application Association**

10.4.7.5 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.

10.4.7.6 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 136.

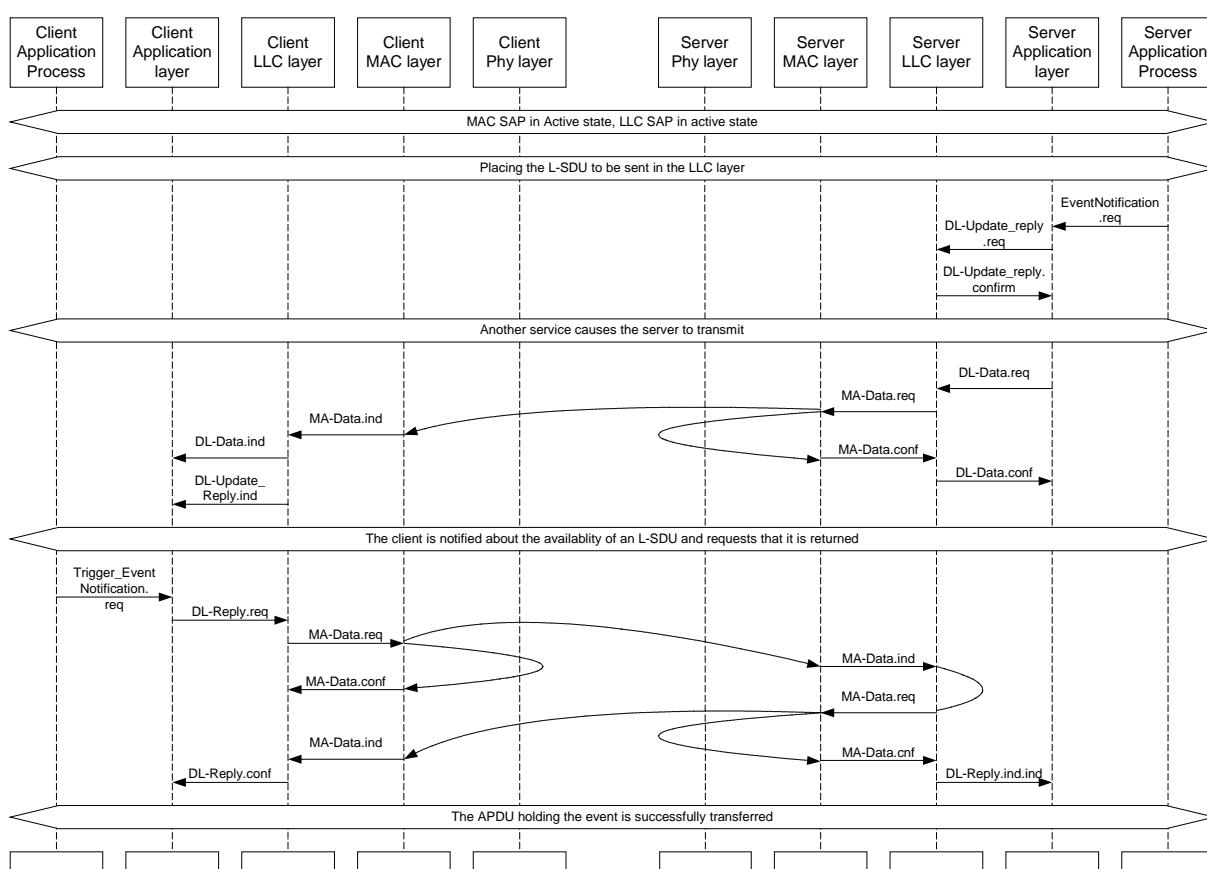


Figure 136 – MSC for an EventNotification service

10.4.7.7 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.

10.4.7.8 Broadcasting

Broadcast messages can be sent by the data concentrator, acting as a client, to servers using broadcast addresses.

10.4.8 Specific considerations / constraints for the HDLC LLC sublayer based profile

10.4.8.1 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 137.

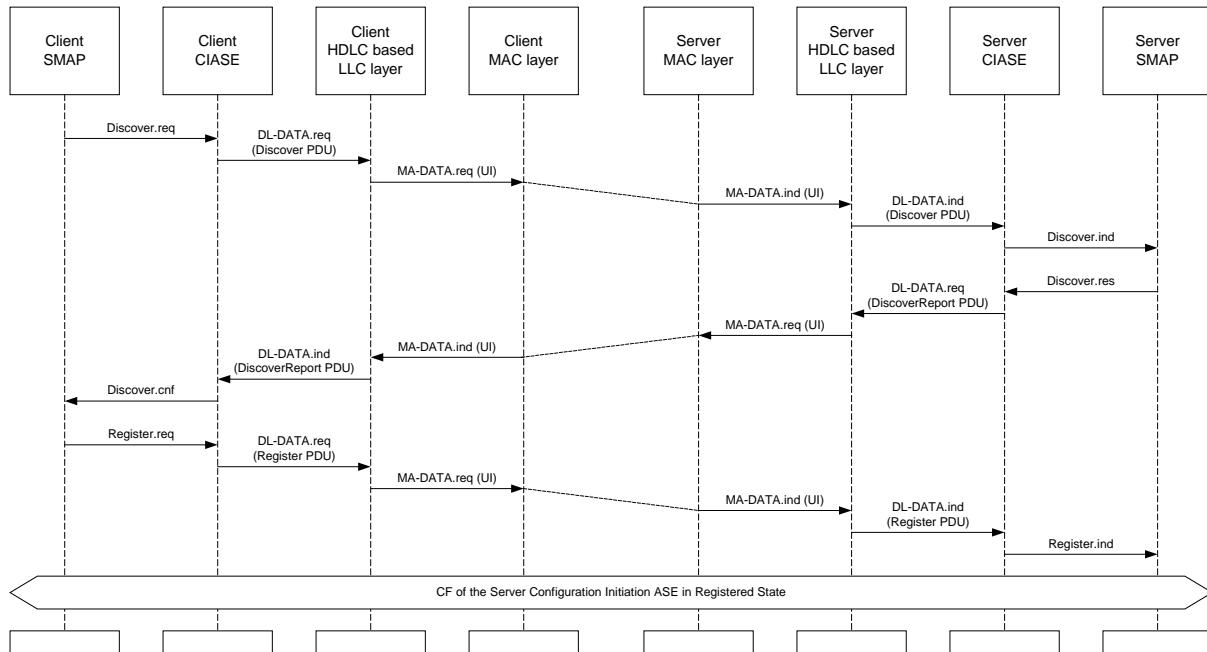


Figure 137 – MSC for the Discovery and Registration process

The MSC for the establishment of a confirmed AA establishment is shown in the upper part of Figure 138.

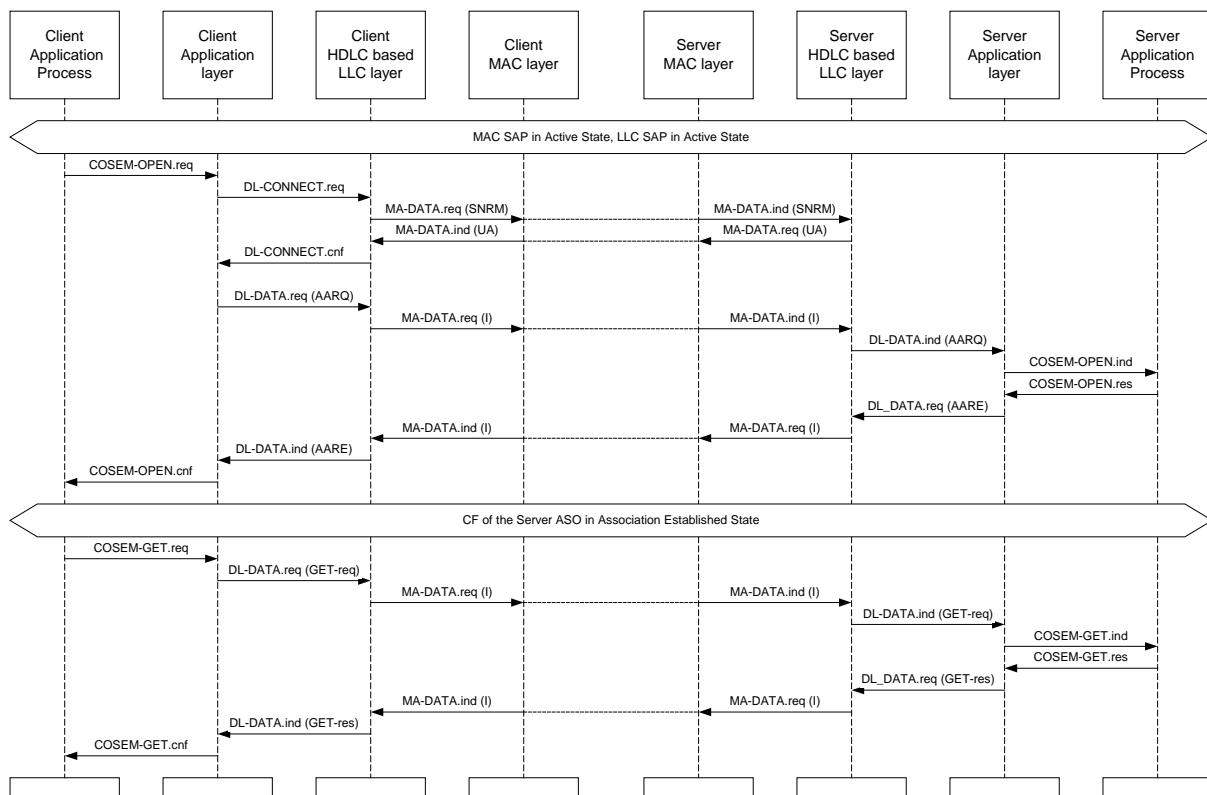


Figure 138 – MSC for successful confirmed AA establishment and the GET service

10.4.8.2 Application association types, confirmed and unconfirmed xDLMS services

10.2.6.1 applies.

10.4.8.3 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 138.

10.4.8.4 Correspondence between AAs and data link layer connections, releasing AAs

10.2.6.2 applies.

10.4.8.5 Service parameters of the COSEM-OPEN/ -RELEASE/ -ABORT services

10.2.6.3 applies.

10.4.8.6 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.

10.4.8.7 Transporting long messages

10.2.6.5 applies.

10.4.8.8 Broadcasting

Broadcast messages can be sent by the data concentrator, acting as a client, to servers using broadcast addresses.

10.4.9 CIASE APDUs

```

CIASEpdu 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),
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	386/500
-----------------------	------------	------------------------	---------

```

-- 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-addr-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),
}

```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	387/500
-----------------------	------------	------------------------	---------

```
    ic-equal-credit-out-of-range      ( 5 ) ,  
    ping-no-response                ( 6 ) ,  
    ping-system-title-nok           ( 7 )  
}
```

END

10.5 The wired and wireless M-Bus profile

10.5.1 Scope

This subclause specifies DLMS/COSEM wired and wireless M-Bus communication profiles for local and neighbourhood networks.

NOTE 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 Technical Report.

NB: The scope of this communication profile 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 Technical Report. Any project specific definitions of data structures and data contents may be provided in project specific companion specifications.

Subclause 10.5.7 provides information on M-Bus frame structures, addressing schemes and an encoding example.

10.5.8 provides MSCs for representative instances of communication.

10.5.2 Targeted communication environments

In the context of the smart metering architecture introduced in IEC 62056-1-0 and shown in Figure 139, 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.

In all cases, metering devices act as DLMS/COSEM servers.

On the C and M interface, metering devices act as M-Bus slaves. The M-Bus master is the NNAP or the LNAP.

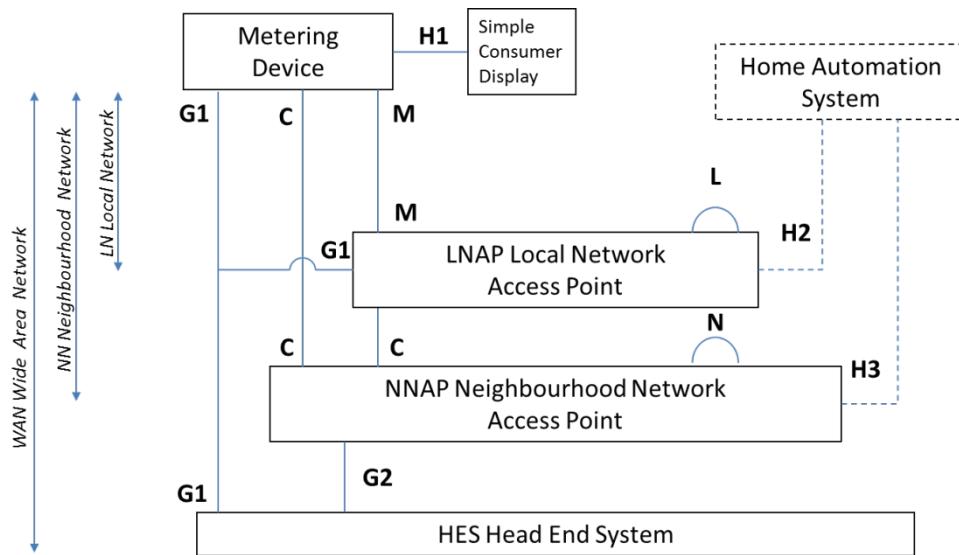


Figure 139 – Entities and interfaces of a smart metering system using the terminology of IEC 62056-1-0

On the H1 and H2 interface the metering device acts as a DLMS/COSEM 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.

10.5.3 Use of the communication layers for this profile

10.5.3.1 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 10.5.3.3 provides additional information on the use of the M-Bus transport layer in this communication profile.

10.5.3.2 Structure of the communication profiles

The structure of the DLMS/COSEM M-Bus wired and wireless communication profiles is shown in Figure 140.

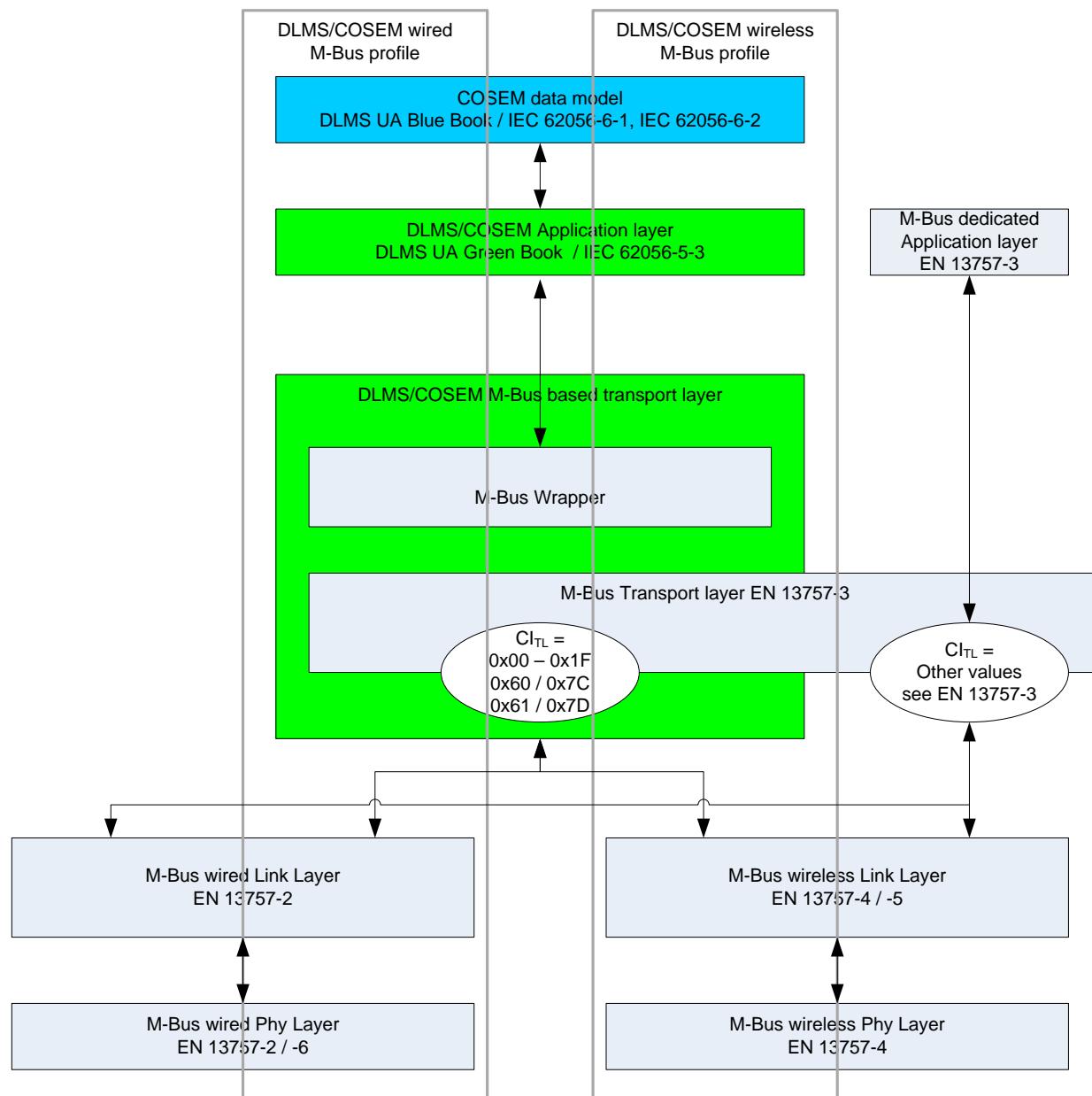


Figure 140 – The DLMS/COSEM wired and wireless M-Bus communication profiles

10.5.3.3 Lower protocol layers and their use

10.5.3.3.1 Physical layer

The physical layer is as specified in EN 13757-2:2004 (wired, twisted pair based) and in EN 13757-4:2013 (wireless).

NOTE For battery operated masters and/or a small number of connected meters a wired M-Bus Physical Layer is specified in EN 13757-6 (twisted pair based for short distances).

10.5.3.3.2 Link layer

The M-Bus link layer is as specified in EN 13757-2:2004 (wired, twisted pair based) and in EN 13757-4:2013 (wireless).

NOTE 1 For wireless meter readout a relaying function may be of interest to extend the range of transmission. EN13757-5:2015 supports simple retransmission (single-hop) and routed wireless networks.

10.5.3.3.3 Transport layer

The M-Bus transport Layer is as specified in EN13757-3:2013.

Together with an M-Bus wrapper specified in 10.5.4.6 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 application layer.

The M-Bus transport layer allows several application layers to co-exist over the M-Bus lower layers. These may 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 application layer is used.

NOTE There are also CI field values for network management purposes. M-Bus frames carrying such CI field values do not necessarily carry application data.

10.5.3.4 Service mapping and adaptation layers

10.5.3.4.1 Overview

As already mentioned in 10.5.3.3.3, 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 application layer.

It comprises the transport layer specified in EN13757-3:2013 and a wrapper layer.

It provides OSI-style connectionless data services with optional segmentation and reassembly to the service user DLMS/COSEM AL.

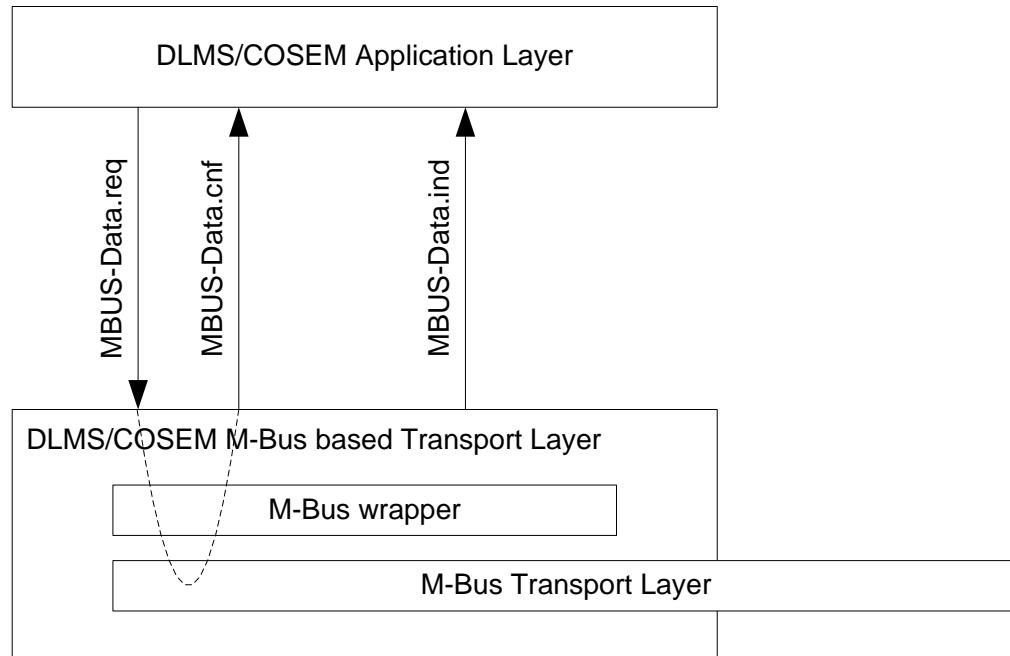
The M-Bus wrapper – specified in 10.5.4.6 – provides addressing capability required to address client and server DLMS/COSEM APs.

The service primitives are shown in Figure 141 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.

**Figure 141 – Summary of DLMS/COSEM M-Bus based TL services****10.5.3.4.2 MBUS-DATA service primitives****10.5.3.4.2.1 MBUS-DATA.request***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 Multi- or broadcasting is available only in the direction client to server.

Semantic

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 M-Bus Data Header type to be used in the M-Bus frame to be sent. Its value may be *None_M-Bus_Data_Header*, *Short_M-Bus_Data_Header* or *Long_M-Bus_Data_Header*, see Figure 144.

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.

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 multi- 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 10.5.3.4.3.

10.5.3.4.2.2 MBUS-DATA.indication

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.

Semantic

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 may be *None_M-Bus_Data_Header*, *Short_M-Bus_Data_Header* or *Long_M-Bus_Data_Header*; see Figure 144.

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.

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_{TL} 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.

10.5.3.4.2.3 MBUS-DATA.confirm

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.

Semantic

The primitive shall provide the service parameters as follows:

```
MBUS-DATA.confirm (
```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	393/500
-----------------------	------------	------------------------	---------

```

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 may be SUCCESS, PENDING or FAILED.

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.

10.5.3.4.3 MBUS-DATA protocol specification

10.5.3.4.3.1 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 144:

- the CI_{TL} field, that indicates the kind of M-Bus Data Header used;
- the M-Bus Data Header, according to the value of the CI_{TL} field;
- the STSAP;
- the DTSAP; and the
- Data i.e. the COSEM APDU or – when segmentation is used – a part of it.

The value of the CI_{TL} 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_{TL} 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_{TL} 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_{TL} field shall be 0x60 in a M-Bus frame sent by a master and 0x7C in a M-Bus frame sent by a slave.

In the case of pull operation, the kind of the 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 push operation – using the xDLMS DataNotification service, see 9.3.10 – the M-Bus Data Header type is determined by the M-Bus slave.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	394/500
-----------------------	------------	------------------------	---------

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.

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.

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 TP to 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 145 – 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.

NOTE The MBUS-Data.conf service primitive may be invoked by the M-Bus based TL with the value == PENDING multiple times while the segments are being sent.

10.5.3.4.3.2 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.

NOTE 1 A STSAP and DTsap pair is valid if there is an AA bound to these SAPs.

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.

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 will have been received.

However, if a segment is missing, all segments shall be discarded and the .confirm service primitive shall not be invoked.

NOTE 2 The M-Bus based TL does not provide a mechanism to recover lost segments.

10.5.3.5 Registration and connection management

Devices hosting DLMS/COSEM 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 Technical Report.

NOTE 1 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 EN13757-3:2013. Specific aspects for wired M-Bus networks are specified in EN 13757-2:2004 and for wireless M-Bus networks in EN 13757-2:2004 and EN13757-5:2015.

The primary_address of the device – see 10.5.4.2 and 10.5.4.3 – is held by the “DLMS/COSEM server M-Bus port setup” object *primary_address* attribute. See DLMS UA 1000-1, 4.8.6.

10.5.4 Identification and addressing scheme

10.5.4.1 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 10.5.4.2, 10.5.4.3 and 10.5.4.4;
- the Transport Layer address CI_{TL} acts as a protocol selector and provides information on the structure of the TPDU and the options / capabilities available. In this profile, it selects the DLMS/COSEM M-Bus based TL. See 10.5.4.5;
- the M-Bus wrapper, contained in the TL, provides addressing for COSEM client and server APs. See 10.5.4.6.

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.

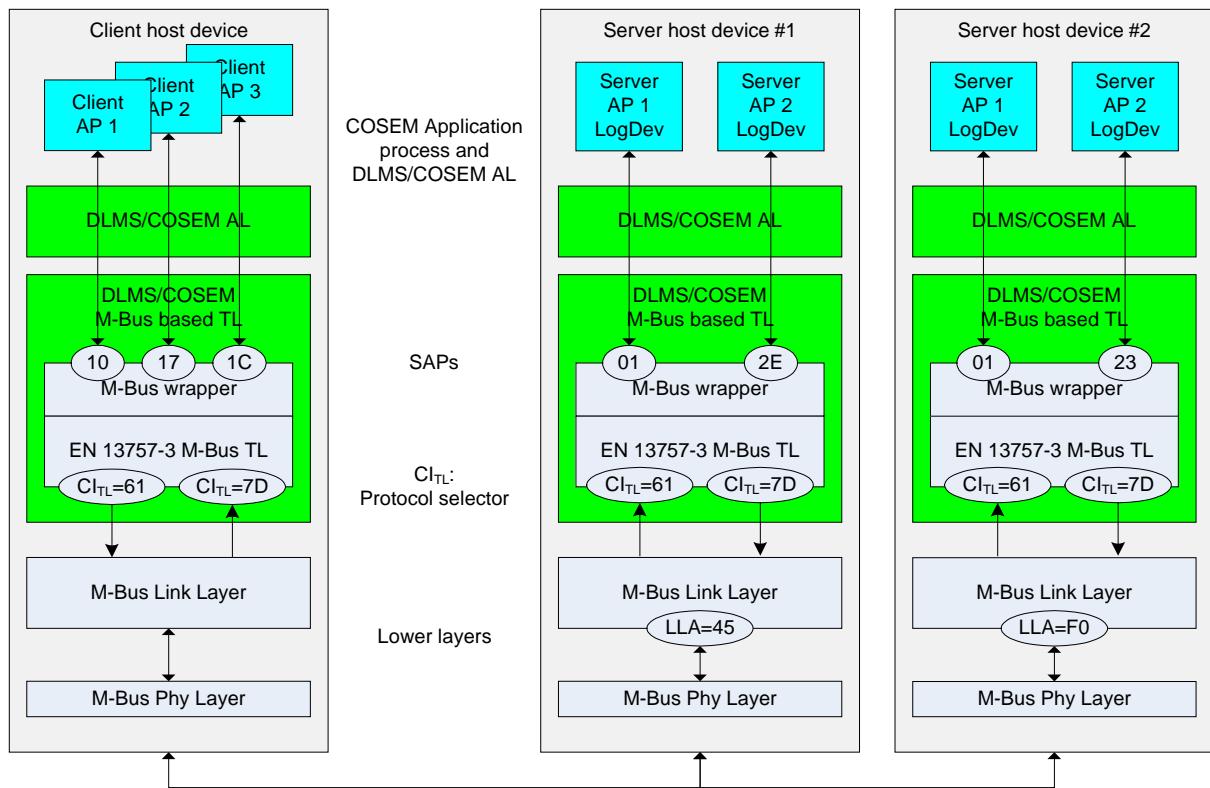
Together, these addresses allow messages to be transported non-ambiguously between a specific client AP and a specific server AP.

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 142.

The triplet {LLA, CI_{TL} , SAP} unambiguously identifies an AP:

- when the client AP#1 (Public Client) wants to send an 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 10.5.4.2.



NOTE All addresses and CI codes are in hexadecimal

LogDev = Logical device

Cl_{TL} values indicate short M-Bus Data Header present

Figure 142 – Identification and addressing scheme in the wired M-Bus profile

10.5.4.2 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 address-less M-Bus Master. The addressing range is as specified in Table 102.

Table 102 – Wired M-Bus Link Layer Addresses

Value	Description
0x00	Un-configured slaves
0x01-0xFA	Primary address ^a range for slaves
0xFB	Reserved for management communication with the primary master repeater
0xFC	Reserved
0xFD	Reserved for secondary addressing ^b
0xFE	Test and Diagnostic address
0xFF	Broadcast address

NOTE Most recent EN13757 standards apply the following terms:

^a Link Layer Address (LLA) instead of Primary address

^b (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 addressing scheme for the Link Layer Address of the wired M-Bus are provided in EN 13757-2:2004, 5.7.5.

10.5.4.3 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 143.

Manufacturer identification (M-ID)	Device Identification No (Serial No.)	Device Version (VER)	Device Type (DT)
------------------------------------	---------------------------------------	----------------------	------------------

Figure 143 – Link Layer Address for wireless M-Bus

If the CI_{ELL} 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_{ELL} indicates usage of the Extended Link Layer with additional control service capabilities.

10.5.4.4 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 address in ALA and in the wireless LLA is 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 EN13757-3:2013, 11.5):

- 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.

10.5.4.5 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 and is introduced by a Control Information (CI) field indicating the kind of the layer and the capabilities provided.

The CI_{TL} field values introducing the DLMS/COSEM M-Bus based TL are specified in Table 103.

Table 103 – DLMS/COSEM M-Bus based TL CI_{TL} values

CI _{TL}	Description
0x00-0x1F	No M-Bus Data Header is present ¹
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

¹ In this case, segmentation / reassembly is possible with restrictions.

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 144.

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..0x0F	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

Figure 144 – 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 10.5.3.4.3).

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 145:

b7	b6	b5	b4	b3	b2	b1	b0
0	0	0	FIN	Sequence number			

Figure 145 – 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 10.5.3.4.3.

The values $CI_{TL} = 0x61 / 0x7D$ 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 $CI_{TL} = 0x60 / 0x7C$ 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 unless when segmentation needs to be used.

Table 104 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 104 – CI fields used for link management purposes

CI field	Upper layers	Description
0x80	Pure TL ^a	Long M-Bus Data Header present, to meter
0x8A	Pure TL ^a	Short M-Bus Data Header present, from meter
0x8B	Pure TL ^a	Long M-Bus Data Header present, from meter
0x8C	ELL ^b	Short M-Bus ELL without ELLA
0x8E	ELL ^b	Long M-Bus ELL with ELLA
^a This CI field values indicate that the message is not intended to an Application layer; no APDU is present.		
^b The Extended Link Layer (ELL) is used only with wireless M-Bus.		

10.5.4.6 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 142 and Figure 144.

The M-Bus wrapper that constitutes a part of the DLMS/COSEM M-Bus based TL provides the required addressing of DLMS/COSEM client and server APs. The values of the SAPs as one byte addresses on the client and the server side are specified in Table 105.

Table 105 – Client and server SAPs

Client SAPs	
No-station	0x00
Client Management Process	0x01
Public Client	0x10
<i>Open for client SAP assignment</i>	0x02...0x0F 0x11... 0xFF
Server SAPs	
No-station	0x00
Management Logical Device	0x01
Reserved for future use	0x02...0x0F
<i>Open for server SAP assignment</i>	0x10...0x7E
All-station (Broadcast)	0xFF

10.5.5 Specific considerations/constraints for using certain services within profile

10.5.5.1 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 battery supply and the device type / configuration shall be considered by the implementer. See also 10.5.5.5.

With respect to efficiency in messaging the following mechanism is applied to perform AA establishment and subsequent message exchange between client and server on Data link Layer of M-Bus:

- M-Bus messages with application data in each message shall follow the scheme acc. to EN 13757-4:2013 where in combination with the SND-UD2 message the control byte code C = 0x43 allows to get data instead of ACK (refer to EN 13757-4:2013 Table 24 – Function codes of the C-field in messages sent from primary stations);
- After AA establishment 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).

10.5.5.2 Application Association establishment and release: ACSE services

Table 106 summarizes the rules for establishing confirmed and unconfirmed AAs and exchanging COSEM APDUs.

Table 106 – 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.

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 AA-s 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.

10.5.5.3 xDLMS services

10.5.5.3.1 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 not allowed 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.

10.5.5.3.2 Unsolicited services

Application of the DLMS/COSEM unsolicited services (EventNotification, DataNotification, InformationReport in M-Bus networks depend on capability and configuration of M-Bus entities and are therefore should be specified in project specific companion specifications.

10.5.5.3.3 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 application level may need to be performed.

In general also broadcast / multicast for secondary addresses is available using the wildcard mechanism in the address parts (EN13757-3:2013, referred by EN 13757-4:2013).

Therefore in wireless M-Bus networks broadcast / multicast messages are applicable, taking into account that from DLMS viewpoint broadcast of COSEM APDUs via wireless M-Bus is available if supporting M-Bus lower layers will offer corresponding functionality.

10.5.5.4 Security mechanisms

The security mechanism specified in 9.2 and DLMS UA 1000-1 can be used without specific constraints.

10.5.5.4.1 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 10.5.3.4.3 and 10.5.4.5. 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.

10.5.5.5 Media access, bandwidth and timing considerations

For DLMS/COSEM unsolicited services – see 10.5.5.3.2 – the M-Bus alarm messaging in wired and wireless M-Bus installations could get certain delays due to:

- out of transmission credits¹⁰ 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/multicast, where EN 13757-4:2013 Table 27 indicates different accessibility (no, temporary, limited, unlimited) with device examples;

¹⁰ The battery operated meter has to protect itself from frequent periodical readout in context with available communication budget. For this constraint such 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 a next credit.

- mains powered meters/communication modules are accessible without limitations (with respect to duty cycles!). On the other hand, battery powered pose some restrictions (see transmission credits), but may operate with dedicated time windows to receive messages.

10.5.6 Communication configuration and management

The COSEM interface classes that apply to the communication profiles specified in this Technical Report are the following:

- DLMS/COSEM server M-Bus port setup (class_id = 76), see DLMS UA 1000-1 Ed. 12.1:2015, 4.8.6;
- M-Bus slave port setup (class_id = 25), see DLMS UA 1000-1 Ed. 12.1:2015, 4.8.2;
- Wireless Mode Q channel (class_id = 73), see DLMS UA 1000-1 Ed. 12.1:2015, 4.8.4;
- M-Bus diagnostic (class_id = 77, version = 0), see DLMS UA 1000-1 Ed. 12.1:2015, 4.8.7.

10.5.7 M-Bus frame structures, addressing schemes and examples

10.5.7.1 General

This Annex 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 10.5.7 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-Fields¹¹ relevant for the DLMS/COSEM M-Bus communication profiles are specified in clause 10.5.4.5.

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 146;
- the physical structure of the M-Bus slave (e.g. with integrated or with external radio unit), see Figure 146;;
- the operation phase: installation or normal operation;
- the installation process, e.g. manual or automatic address assignment.

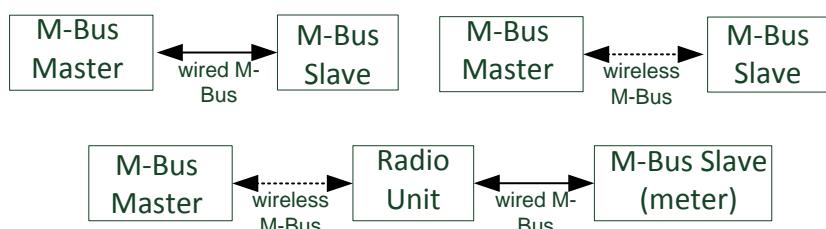


Figure 146 – M-Bus communication paths direct or cascaded

In the following sections M-Bus frame structures and addressing related information are described and examples are shown for:

- Wired M-Bus:

Figure 147 – Wired M-Bus frame structure, none M-Bus Data Header;

¹¹ Further CI-Fields are specified for management purposes, see EN 13757-3:2013 and EN 13757-4:2013.

Figure 148 – Wired M-Bus frame structure with long M-Bus Data Header;

- Wireless M-Bus:

Figure 149 – Wireless M-Bus frame structure with short ELL, none M-Bus Data Header;

Figure 150 – Wireless M-Bus frame structure with long ELL, none M-Bus Data Header;

Figure 151 – Wireless M-Bus frame structure with long ELL and long M-Bus Data Header.

10.5.7.2 None, Short or Long M-Bus Data Header

10.5.7.2.1 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 102 – 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 worldwide unique (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. This wildcard can be used to search (test) for Slaves connected to the M-Bus (refer to Wild Card Search in EN13757-3:2013). 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 simple wrapper in the Slave response will be sufficient. The Master should in general apply frames with simple wrapper.

NOTE There will be the option to disable the long TL after the address assignment process.

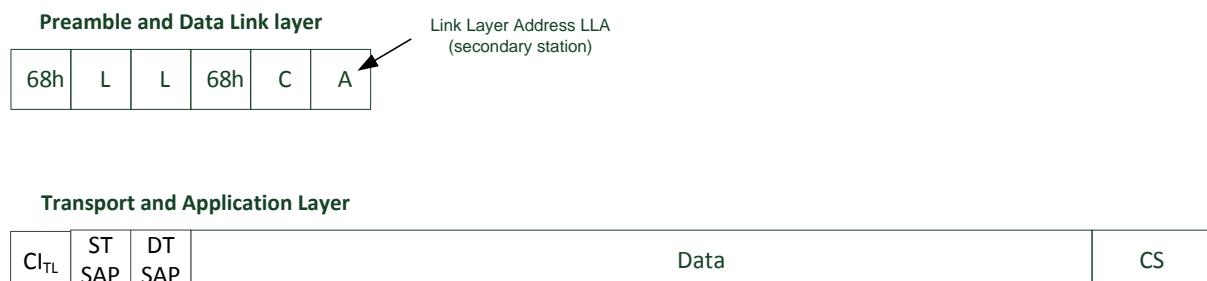
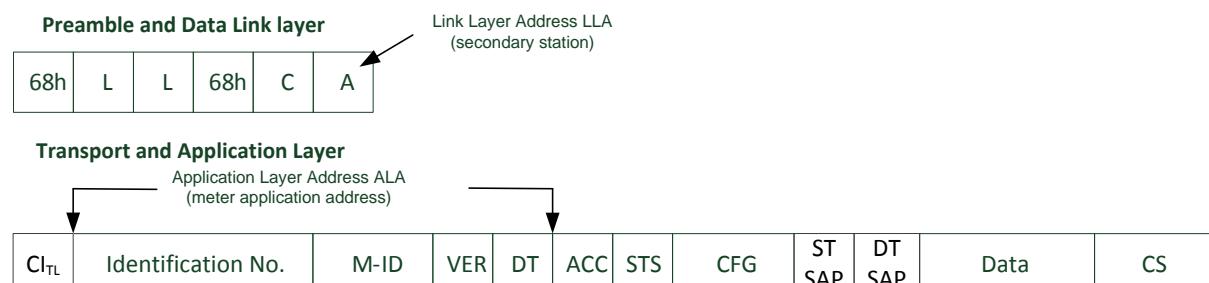
Figure 147 and Figure 148 show wired M-Bus frame structures according to EN 13757-2:2004 / IEC 60870-5-1:1990.

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 _{TL}	CI-Field for Transport Layer
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 147 – Wired M-Bus frame structure, none M-Bus Data Header****Figure 148 – Wired M-Bus frame structure with long M-Bus Data Header**

NOTE 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.

10.5.7.3 Wireless M-Bus**10.5.7.3.1 Extended Link Layer**

For the communication via wireless M-Bus the short or long Extended Link Layer shall be used in each message transmitted.

NOTE The wireless M-Bus uses the Access number ACC for the identification of an old or new M-Bus frame types, see EN13757-3:2013, 5.9. Therefore, the wireless M-Bus protocol must 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 not requested messages. The master may access the slave after such a transmission. For all frames transmitted between master and the server the LLA and the ELLA shall be applied. For the not requested transmission of the slave only the LLA is required.

Figure 149, Figure 150 and Figure 151 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 Transport Layer 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

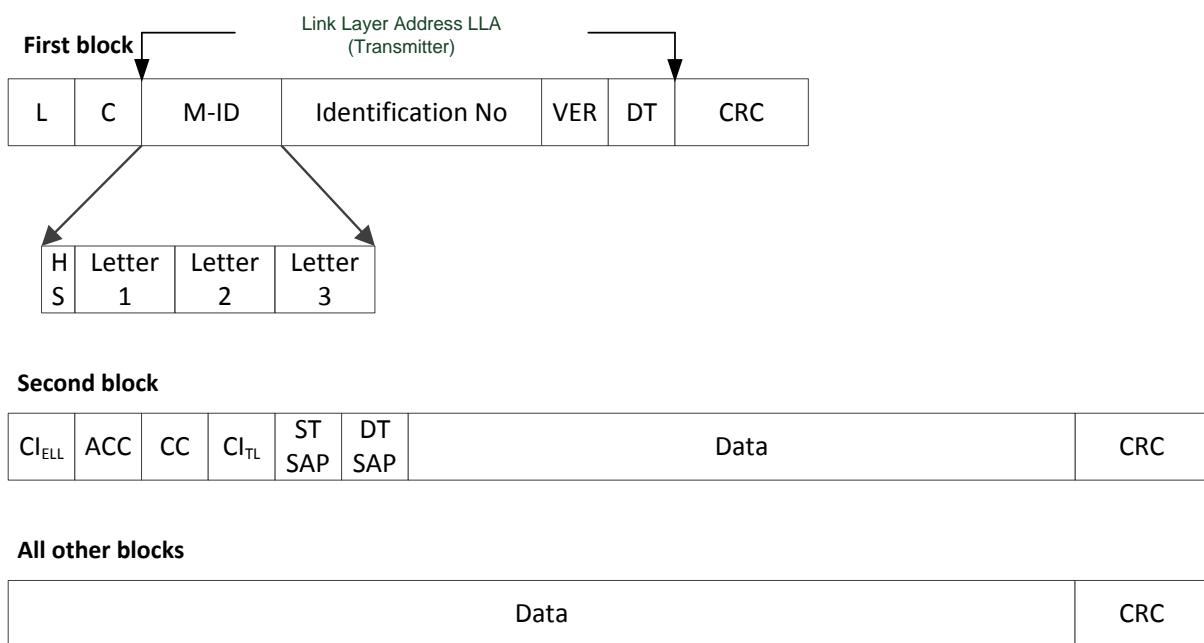
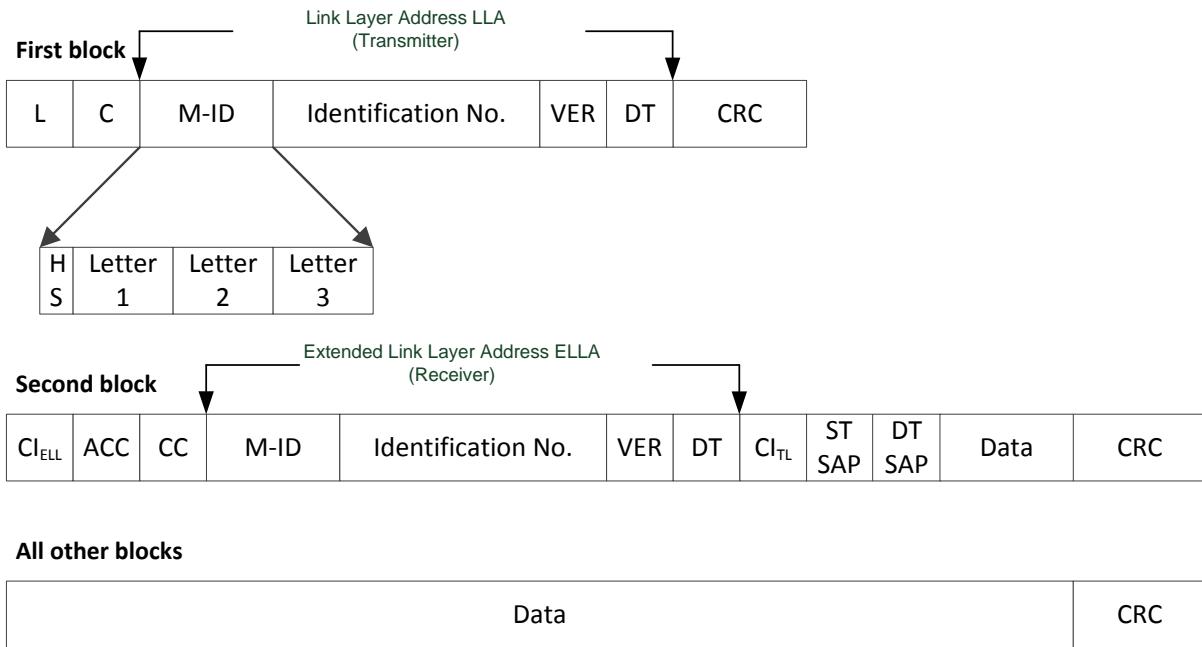
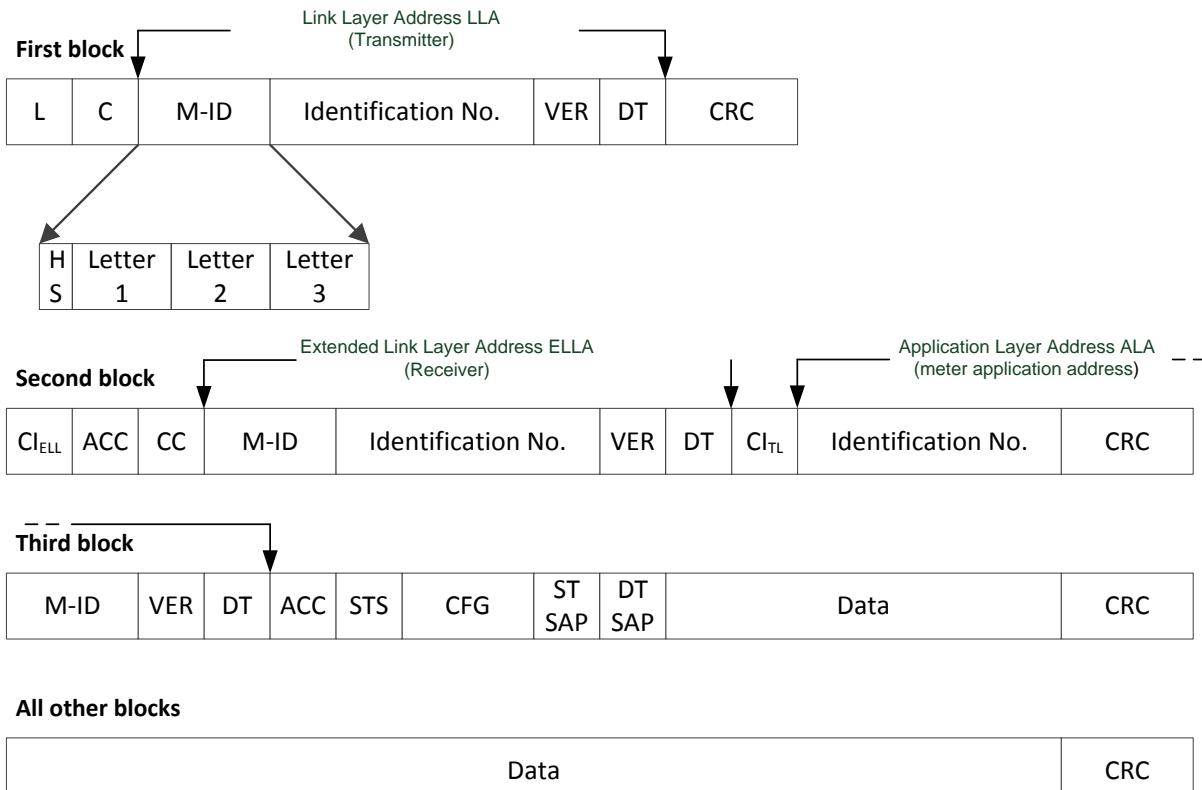


Figure 149 – Wireless M-Bus frame structure with short ELL, none M-Bus Data Header

**Figure 150 – Wireless M-Bus frame structure with long ELL, none M-Bus Data Header****Figure 151 – Wireless M-Bus frame structure with long ELL and long M-Bus Data Header**

NOTE The Link Layer Address (LLA) contains always the transmitter address (radio unit). The Extended Link Layer Address (ELLA) contains always the receiver address.

The ELLA is available only in frames with a long Extended Link Layer.

The Application Layer Address (ALA) contains always the meter application address (M-Bus address of the Slave entity). The Application Layer Address is available only in frames with CI-Fields indicating an M-Bus long Data Header.

Frames without or with short Transport Layer are applied only, when the Link Layer Address is sufficient.

The Link Layer address (LLA+ELLA) is a hard coded address assigned by the manufacturer and shall not be changeable for the operator. In case of an external communication adapter the Application Layer Address need to be assigned to this adapter.

10.5.7.3.2 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 Application Layer Address contains the intended Slave Address.

Otherwise the wrapper with the CI-Field value 0x10 may be used (but no segmentation).

10.5.7.4 Encoding example: Data-Notification carrying daily billing data

10.5.7.4.1 Overview

This 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 separating the metadata from the data. This drastically reduces the overhead which is essential for battery operated devices and for restrained communication media like wireless M-Bus.

The example also shows the encoding of the information related to the lower layers.

10.5.7.4.2 Example: Daily billing data

Table 107 shows the daily billing data that are captured – together with the *template_id* – to the *compact_buffer* attribute of a “Compact data” object.

Table 107 – 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 152 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				Number of octets							
82				113							
Payload size as compact data = 34 octets											
Application Layer											
TL											
Data Link Layer (DLL)											
Extended Link Layer (ELL)											
Layer											
Encrypted APDU											

Figure 152 – Daily billing data without / with DLMS/COSEM security applied

10.5.8 Message sequence charts

This subclause 10.5.8 shows some message sequence charts for communication between a DLMS/COSEM 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 153 shows the MSC for the COSEM-OPEN service for wired M-Bus without M-Bus header.

Figure 154 shows the MSC for the GET service for wired M-Bus without M-Bus header.

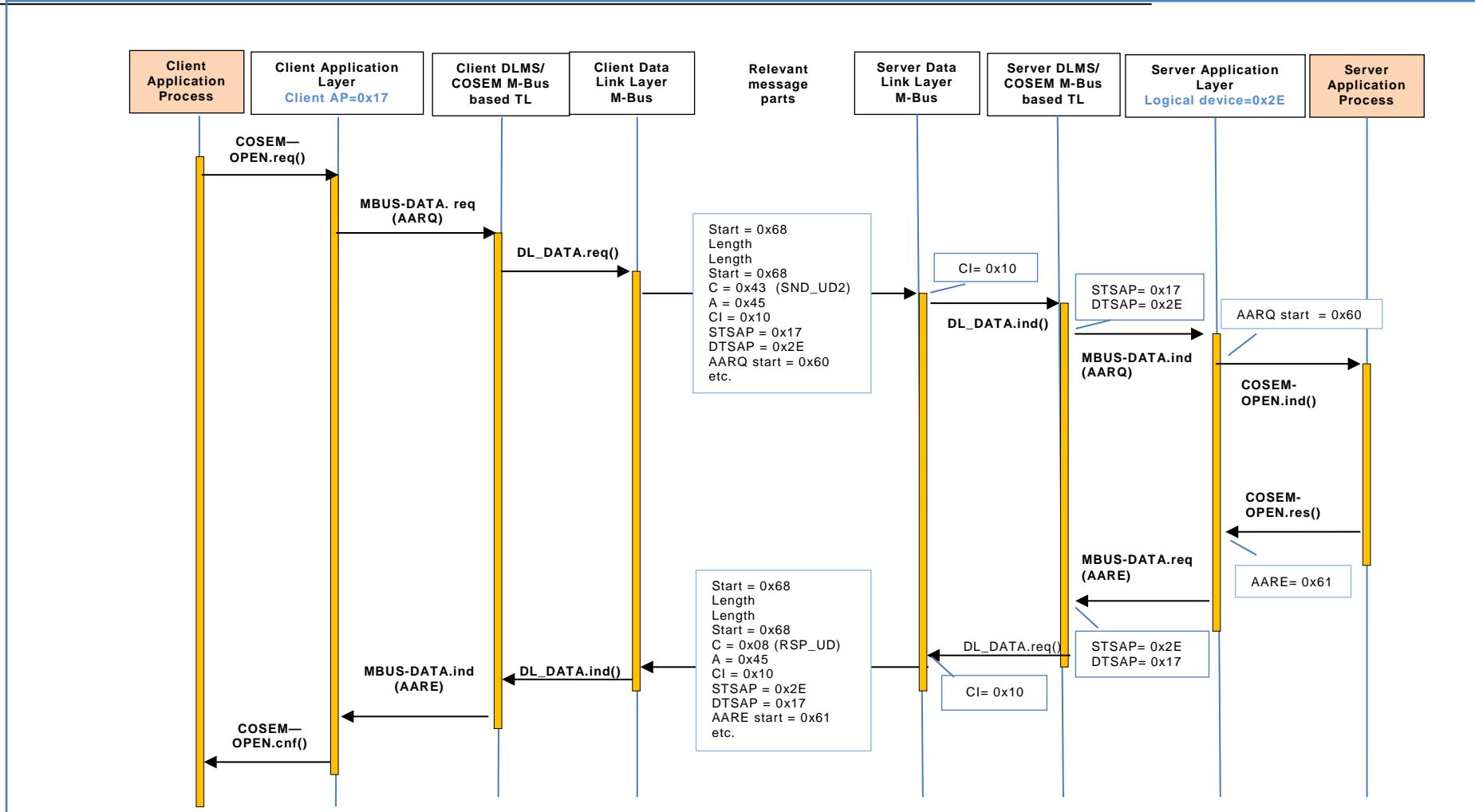


Figure 153 – MSC for the COSEM-OPEN service for wired M-Bus, none M-Bus header

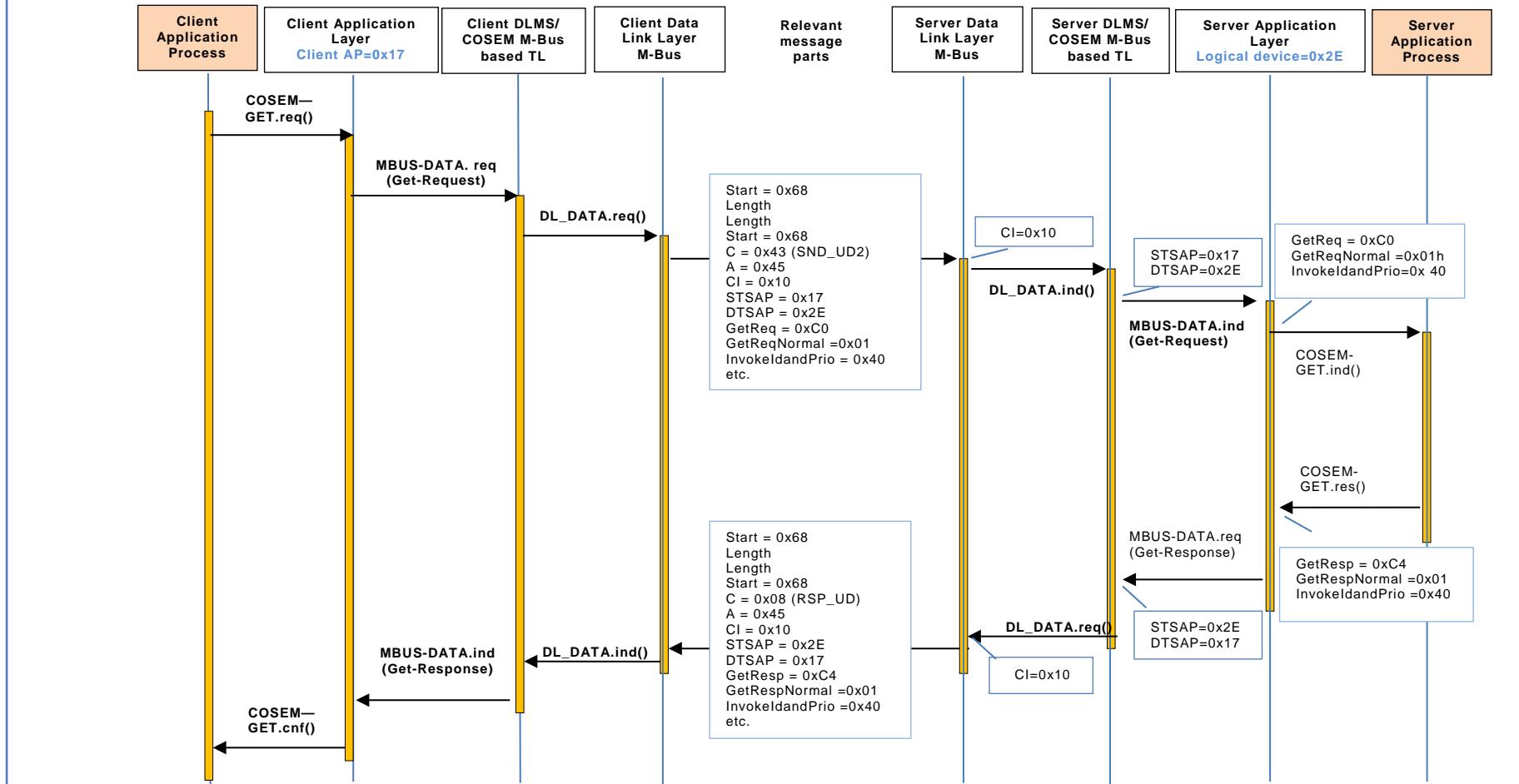


Figure 154 – MSC the GET service for wired M-Bus, none M-Bus header

10.6 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 155:



Where:

- Dst_AP = Destination AP identifies the destination Application Process;
- Src_AP = Source AP identifies the source Application Process.

Figure 155 – Short wrapper

Table 108 specifies the identifiers of reserved Application Processes:

Table 108 – Reserved Application Process SAPs

Client side reserved SAPs	
No-station	0x00
Client Management Process	0x01
Public Client	0x10
<i>Open for client AP assignment</i>	0x02...0xF
	0x11 and up
Server side reserved SAPs	
No-station	0x00
Management Logical Device	0x01
Reserved	0x02...0xF
<i>Open for server SAP assignment</i>	0x10...0x7E
All-station (Broadcast)	0x7F

10.7 Gateway protocol

10.7.1 General

This subclause 10.7 specifies a method for exchanging data between DLMS/COSEM 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/COSEM 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 meter or in a stand-alone device.

The DLMS/COSEM specification for meter 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 meter 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.

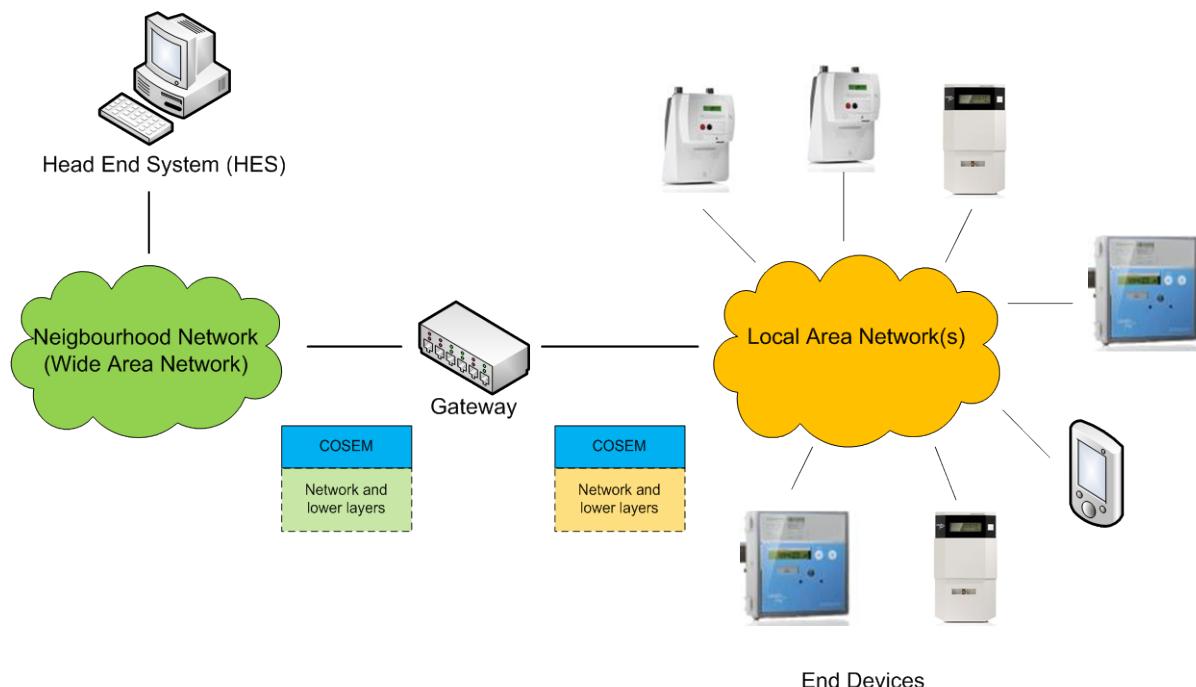


Figure 156 – General architecture with gateway

The DLMS/COSEM client (HES) reaches the gateway via a WAN or via NNAP; see Figure 156. The gateway itself may be a stand-alone device or a DLMS/COSEM meter 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).

10.7.2 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 157.

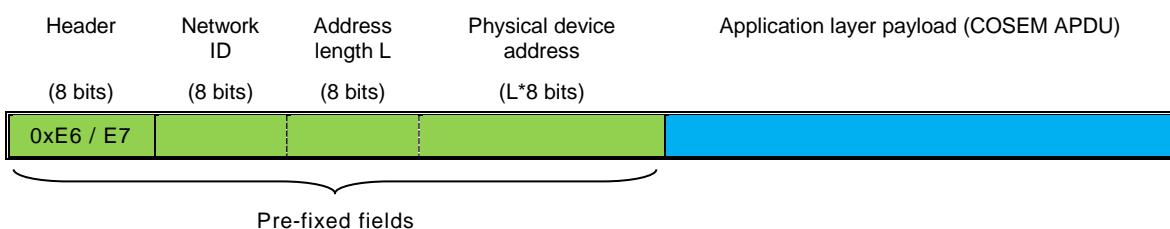


Figure 157 – The fields used for pre-fixing the COSEM APDUs

- 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/COSEM client to a DLMS/COSEM server or a request message from a DLMS/COSEM server to a DLMS/COSEM client (data notification);
 - 0xE7 indicates a response from the DLMS/COSEM server to the DLMS/COSEM 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 meter, the pre-fixed fields are not present.

10.7.3 HES in the WAN/NN acting as Initiator (Pull operation)

In the sequence diagram shown in Figure 158 the traditional pull data exchange between the DLMS/COSEM client and server via a gateway is further elaborated:

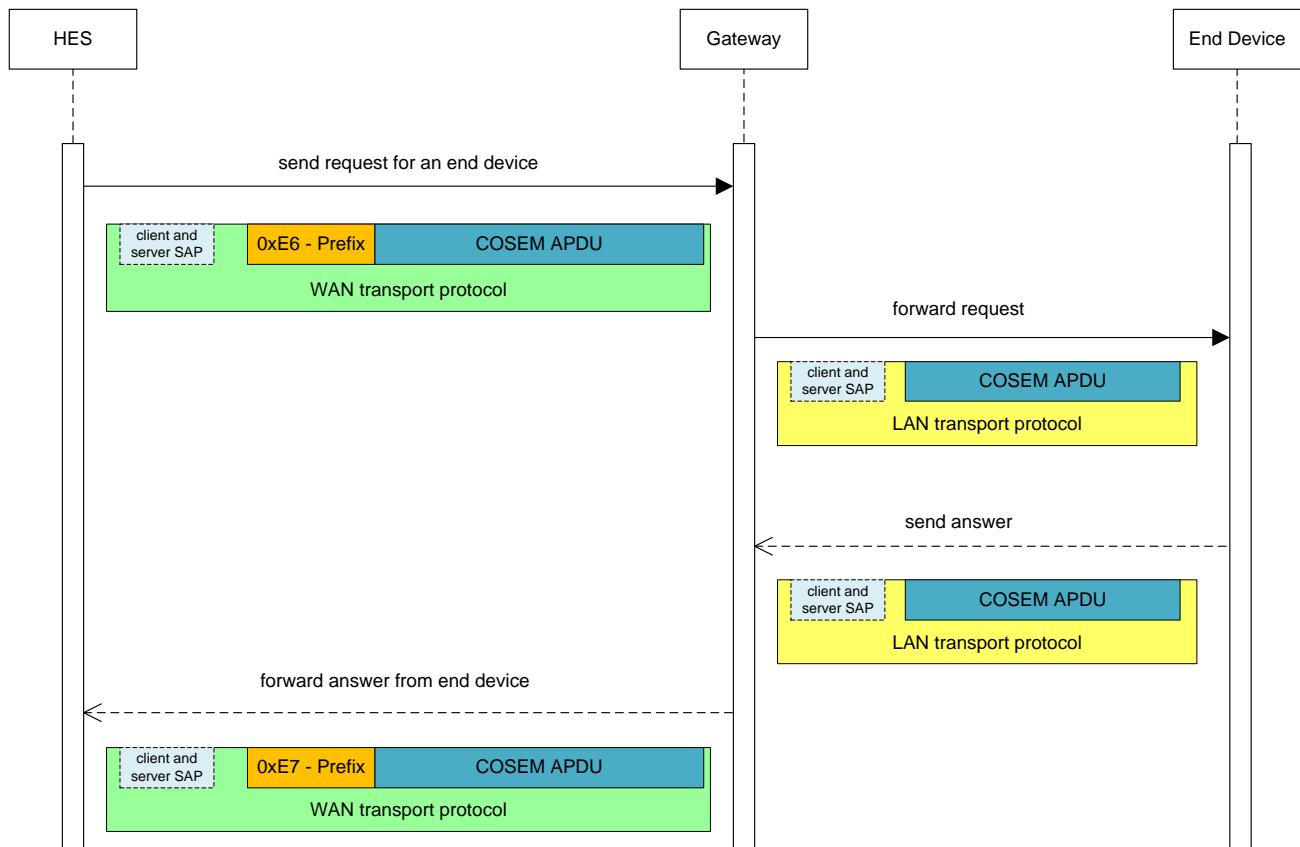


Figure 158 – Pull message sequence chart

Prerequisite: The client in the WAN/NN must know the network ID, the protocol and the physical device address of the server it wants to reach in the LAN.

The DLMS/COSEM client (HES) sends every request, carried by a COSEM APDU, prefixed with four fields as shown in Figure 157 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 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 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.

10.7.4 End devices in the LAN acting as Initiators (Push operation)

10.7.4.1 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.

10.7.4.2 End device with WAN/NN knowledge

Prerequisite: The server in the LAN must know the network ID, the protocol and the address of the client it wants to reach in the WAN/NN.

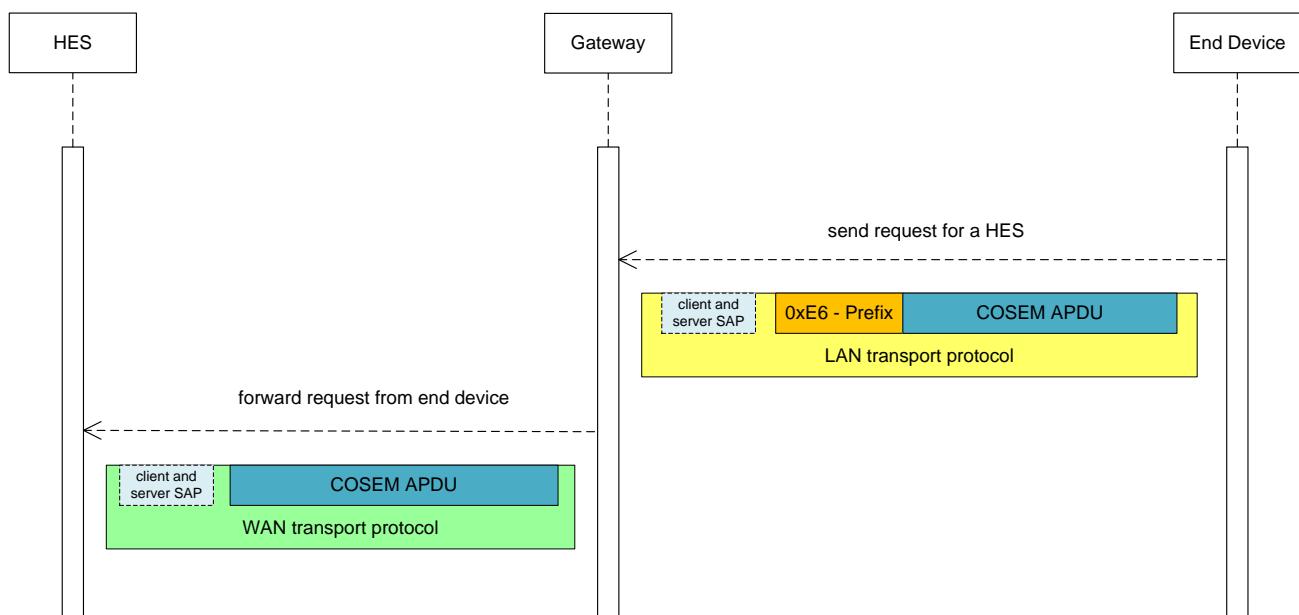


Figure 159 – Push message sequence chart

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 159.

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 layer of the DLMS/COSEM AL on the WAN/NN.

10.7.4.3 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 doesn't require a protocol extension, this use case is not described any further.

10.7.5 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 11 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 9.4.2 – shall be encoded in BER (ISO/IEC 8825). 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 109.

Table 109 – Conformance block

Conformance ::= [APPLICATION 31] IMPLICIT BIT STRING (SIZE(24))		LN referencing		SN referencing	
		Used with	Proposed	Negotiated	Proposed
-- the bit is set when the corresponding service or functionality is available					
reserved-zero (0),			0	0	0
reserved-one (1),			0	0	0
reserved-two (2),			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 110:

Table 110 – A-XDR encoding the xDLMS InitiateRequest APDU

-- A-XDR encoding the xDLMS InitiateRequest APDU	LN referencing	SN referencing
// encoding of the tag of the xDLMS APDU CHOICE (InitiateRequest)	01	01
-- encoding of the dedicated-key component (OCTET STRING OPTIONAL)		
// usage flag(FALSE , not present)	00	00
-- encoding of the response-allowed component (BOOLEAN DEFAULT TRUE)		
// usage flag(FALSE , default value TRUE conveyed)	00	00
-- encoding of the proposed-quality-of-service component ([0] IMPLICIT Integer8 OPTIONAL)		
// usage flag(FALSE , not present)	00	00
-- encoding of the proposed-dlms-version-number component (Unsigned8)		
// value= 6, the encoding of an Unsigned8 is its value	06	06
-- encoding of the proposed-conformance component (Conformance , [APPLICATION 31] IMPLICIT BIT STRING (SIZE(24)) ¹)		
// encoding of the [APPLICATION 31] tag (ASN.1 explicit tag) ²	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 (Unsigned16)		
// 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

¹ 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.

² For encoding of identifier octets, see ISO/IEC 8825 Ed.3.0:2002, 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 111.

Table 111 – 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] IMPLICIT Integer8 OPTIONAL)		
// usage flag(FALSE , 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] IMPLICIT BIT STRING (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
    {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);

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	422/500
-----------------------	------------	------------------------	---------

- 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 $40^*\text{First} + \text{Second} \rightarrow 40^*2 + 16 = 96 = 0x60$;
 - 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 112. See also Table 113.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	423/500
-----------------------	------------	------------------------	---------

Table 112 – 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], IMPLICIT BIT STRING { version1 (0) } DEFAULT { version1 })						
// no encoding, thus it is considered with its DEFAULT value						
-- encoding the fields of the Kernel						
-- application-context-name field ([1], Application-context-name, OBJECT IDENTIFIER)						
// 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 (OBJECT IDENTIFIER , 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, BIT STRING { authentication (0) })						
// encoding of the tag of the acse-requirements field ([10], IMPLICIT , 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 BIT STRING	-	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], IMPLICIT Mechanism-name OBJECT IDENTIFIER)						
// encoding of the tag ([11], IMPLICIT , Context-specific)	-	8B	8B	-	8B	8B

DLMS User Association, DLMS/COSEM Architecture and Protocols, Edition 8.1

<i>-- BER encoding the AARQ APDU</i>	LN referencing			SN referencing		
	no sec.	LLS	HLS	no sec.	LLS	HLS
// encoding of the length of the tagged component's value field	-	07	07	-	07	07
// encoding of the value of the OBJECT IDENTIFIER :	-	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
-- calling-authentication-value field ([12], Authentication-value CHOICE)						
// encoding of the tag ([12], EXPLICIT , 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 (<i>charstring [0] IMPLICIT GraphicString</i>)	-	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:	-	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
-- encoding the user-information field component (Association-information, OCTET STRING)						
// 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 (OCTET STRING , Universal)	04	04	04	04	04	04
// encoding of the length of the OCTET STRING '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 113 – 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 meter uses LN referencing, SN referencing is proposed;
 - when the meter 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 114. See also Table 115.

Table 114 – 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], IMPLICIT BIT STRING {version1 (0)} DEFAULT {version1})						
// no encoding, thus it is considered with its DEFAULT value						
-- application-context-name field ([1], Application-context-name, OBJECT IDENTIFIER)						
// 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 (OBJECT IDENTIFIER , 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, INTEGER)						
// 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 (INTEGER , Universal)	02				02	
// encoding of the length of the result's value field	01				01	
// encoding of the value of the Result:	00	01	01	00	00	00

-- 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
// success: 0, accepted							
// failure case 1 and 2: 1, rejected-permanent							
-- result-source-diagnostic field ([3], Associate-source-diagnostic, CHOICE)							
// 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 (INTEGER, 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;	00	02	01	0E	00	0E	
- failure 1: 2, application-context-name not supported;							
- failure 2: 1, no-reason-given.							
- success, HLS security (5): 14, authentication required;							
-- encoding the fields of the authentication functional unit							
-- responder-acse-requirements field ([8], IMPLICIT , ACSE-requirements, BIT STRING { authentication (0) })							
// encoding of the tag of the acse-requirements field ([8], IMPLICIT , Context-specific)	-			88	-	88	
// encoding of the length of the tagged component's value field.	-			02	-	02	
// encoding of the number of unused bits in the last byte of the BIT STRING	-			07	-	07	

-- 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 authentication functional unit (0)	-				80	-	80
-- mechanism-name field ([9], IMPLICIT , Mechanism-name OBJECT IDENTIFIER)							
// encoding of the tag ([9], IMPLICIT , 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], EXPLICIT , 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] IMPLICIT 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, OCTET STRING)							
// encoding the tag for the user-information field component ([30], Context-specific, Constructed)	BE	BE	BE	BE	BE	BE	BE
// encoding of the length of the tagged component's value field	10	10	06	10	10	10	10
// encoding of the choice for user-information (OCTET STRING , Universal)	04	04	04	04	04	04	04
// encoding of the length of the OCTET STRING's value field	0E	0E	04	0E	0E	0E	0E

<i>-- BER encoding the AARE APDU</i>	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
// 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 115 – 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 the 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 116.

Table 116 – A-XDR encoding of the xDLMS InitiateRequest APDU

// encoding of the tag of the xDLMS APDU CHOICE (<i>InitiateRequest</i>)	01
-- encoding of the dedicated-key component (OCTET STRING OPTIONAL)	
// usage flag (TRUE , present)	01
// length of the OCTET STRING	10
// contents of the OCTET STRING	0011223344556677 8899AABBCCDDEEFF
-- encoding of the response-allowed component (BOOLEAN DEFAULT TRUE)	
// usage flag (FALSE , default value TRUE conveyed)	00
-- encoding of the proposed-quality-of-service component ([0] IMPLICIT Integer8 OPTIONAL)	
// usage flag (FALSE , not present)	00
-- encoding of the proposed-dlms-version-number component (Unsigned8)	
// value = 6; the A-XDR encoding of an Unsigned8 is its value	06
-- encoding of the proposed-conformance component (Conformance, [APPLICATION 31] IMPLICIT BIT STRING (SIZE(24))) ¹	
// encoding of the [APPLICATION 31] tag (ASN.1 explicit tag) ²	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 (Unsigned16)	
// value = 0x04B0, the encoding of an Unsigned16 is its value	04B0
-- resulting octet-string	0101100011223344 5566778899AABBCC DDEEFF0000065F1F 0400007E1F04B0

¹ 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.

² For encoding of identifier octets, see ISO/IEC 8825 Ed.3.0:2002, 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 117 shows the encoding of an xDLMS InitiateRequest APDU which is also authenticated and encrypted.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	432/500
-----------------------	------------	------------------------	---------

Table 117 – Authenticated encryption of the xDLMS InitiateRequest APDU

	<i>X</i>	Contents	LEN(X) bytes	len(X) bits
Security material				
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>FC</i>	01234567	4	32
Initialization Vector	<i>IV</i>	<i>Sys-T</i> <i>FC</i> 4D4D4D0000BC614E01234567	12	96
Block cipher key (global)	<i>EK</i>	000102030405060708090A0B0C0D0EOF	16	128
Authentication Key	<i>AK</i>	D0D1D2D3D4D5D6D7D8D9DABDCDDDEF	16	128
Security applied		Authenticated encryption		
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>FC</i> 3001234567	5	40
Inputs				
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> 30D0D1D2D3D4D5D6D7D8D9DABDCDDDEF	17	136
Outputs				
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 118.

Table 118 – 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], IMPLICIT BIT STRING { version 1 (0) } DEFAULT { version1 })	
// no encoding, thus it is considered with its DEFAULT value	
-- encoding of the fields of the Kernel	
-- application-context-name field ([1], Application-context-name, OBJECT IDENTIFIER)	
// 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 (OBJECT IDENTIFIER , 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, BIT STRING { authentication (0) })	
// encoding of the tag of the acse-requirements field ([10], IMPLICIT , 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 BIT STRING	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], IMPLICIT Mechanism-name OBJECT IDENTIFIER)	
// encoding of the tag [11], IMPLICIT , Context-specific)	8B
// encoding of the length of the tagged component's value field	07
// encoding of the value of the OBJECT IDENTIFIER : - low-level-security-mechanism-name,	60857405080201
-- calling-authentication-value field ([12], Authentication-value CHOICE)	
// encoding of the tag ([12], EXPLICIT , Context-specific)	AC
// encoding of the length of the tagged component's value field	0A
// encoding of the choice for Authentication-value (charstring [0] IMPLICIT GraphicString)	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, 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 (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 is shown in Table 119.

Table 119 – A-XDR encoding of the xDLMS InitiateResponse APDU

// encoding of the tag of the xDLMS APDU CHOICE (InitiateResponse)	08
-- encoding of the negotiated-quality-of-service component ([0] IMPLICIT Integer8 OPTIONAL)	
// usage flag (FALSE , not present)	00
-- encoding of the negotiated-dlms-version-number component (Unsigned8)	
// value = 6, the A-XDR encoding of an Unsigned8 is its value	06
-- encoding of the negotiated-conformance component (Conformance, [APPLICATION 31] IMPLICIT BIT STRING (SIZE(24)))	
// 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 (Unsigned16)	
// value = 0x0400, the encoding of an Unsigned16 is its value	0400
-- encoding of the VAA-Name component (ObjectName, Integer16)	
// 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	0800065F1F04000007 7C1F04000007

12.5 Authenticated encryption of the xDLMS InitiateResponse APDU

Table 120 shows the encoding of the xDLMS InitiateResponse APDU which is also authenticated and encrypted.

Table 120 – Authenticated encryption of the xDLMS InitiateResponse APDU

	<i>X</i>	Contents	LEN(X) bytes	len(X) bits
Security material				
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
Frame Counter	<i>FC</i>	01234567	4	32
Initialization Vector	<i>IV</i>	<i>Sys-T</i> <i>FC</i>		
		4D4D4D0000BC614E01234567	12	96
Block cipher key (global)	<i>EK</i>	000102030405060708090A0B0C0D0EOF	16	128
Authentication Key	<i>AK</i>	D0D1D2D3D4D5D6D7D8D9DADBCDDDEDF	16	128
Security applied		Authenticated encryption		
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>FC</i>		
		3001234567	5	40
Inputs				
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> 30D0D1D2D3D4D5D6D7D8D9DADBCDDDEDF	17	136
Outputs				
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 121.

Table 121 – 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	
-- encoding of the fields of the Kernel	
-- application-context-name field ([1], Application-context-name, OBJECT IDENTIFIER)	
// 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 (OBJECT IDENTIFIER , 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, INTEGER)	
// 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 (INTEGER , 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, CHOICE)	
// 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 (INTEGER , 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

-- BER encoding of the AARE APDU	
-- 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, 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 (OCTET STRING , Universal)	04
// encoding of the length of the OCTET STRING's value field	21
-- 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 122.

Table 122 – 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 RLRQ APDU is shown inTable 123.

Table 123 – 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 (OCTET STRING, Universal)	04
// encoding of the length of the OCTET STRING's 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 IEC 61334-4-32:1996 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
```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	440/500
-----------------------	------------	------------------------	---------

01 // Alarm-Descriptor

Register service: MAC frame carrying a Register CI-PDU

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 0F 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 0F 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 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, 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 = FFFF
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

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	441/500
-----------------------	------------	------------------------	---------

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 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 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}
```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	442/500
-----------------------	------------	------------------------	---------

```
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:

```
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 1C
88 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 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
```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	443/500
-----------------------	------------	------------------------	---------

```

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_Adj_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 61334-4-32:1996:1996 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
```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	446/500
-----------------------	------------	------------------------	---------

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 // padding
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
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 // padding
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 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 // 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 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 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 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 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 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
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 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 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
```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	449/500
-----------------------	------------	------------------------	---------

```

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 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 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 01 08
0020  04 00 00 00 01 5F 75 7E 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
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 // pad
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 Alarm-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 125 – Table 132 show examples for data exchange using xDLMS services with LN referencing (left column) and SN referencing (right column). Table 124 shows the objects used in the examples.

Table 124 – The objects used in the examples

Object 1:
- Class: Data
- Logical name: 0000800000FF
- Short name of value attribute: 0100
- Value: octet string of 50 elements
- 01020304050607080910111213141516
- 17181920212223242526272829303132
- 33343536373839404142434445464748
- 4950
Object 2:
- Class: Data
- Logical name: 0000800100FF
- Short name of value attribute: 0110
- Value: visible string of 3 elements 303030

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 125 – 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><="" td=""><td>0501 020100 <ReadRequest Qty="0001" > <VariableName Value="0100" /> </ReadRequest></td></br><>	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><="" td=""><td>0C01 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 <ReadResponse Qty="0001" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> </ReadResponse></td></br><>	0C01 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 <ReadResponse Qty="0001" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> </ReadResponse>

Table 126 – Example: Reading the value of a list of attributes without block transfer

C003C1 02 0001000800000FF0200 0001000800100FF0200 <GetRequestWithList> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptorList Qty="0002" >	0502 020100 020110 <ReadRequest Qty="0002" > <VariableName Value="0100" /> <VariableName Value="0110" /> </ReadRequest>
---	---

<pre> <_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> </pre>	
<p>C403C1 02 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 00 0A03 303030</p> <pre> <GetResponse> <GetResponseWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> <Data> <VisibleString Value="303030" /> </Data> </Result> </GetResponseWithList> </GetResponse> </pre>	<p>0C02 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 00 0A03 303030</p> <p><ReadResponse Qty="0002" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> <Data> <VisibleString Value="303030" /> </Data> </ReadResponse></p>

Table 127 – Example: Reading the value of a single attribute with block transfer

C001C1	0501
--------	------

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	453/500
-----------------------	------------	------------------------	---------

DLMS User Association, DLMS/COSEM Architecture and Protocols, Edition 8.1

<pre>000100080000FF0200 <GetRequest> <GetRequestNormal> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="00080000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </GetRequestNormal> </GetRequest></pre>	<pre>020100 <ReadRequest Qty="0001" > <VariableName Value="0100" /> </ReadRequest></pre>
<pre>C402C1 00 00000001 00 1E 093201020304050607080910111213 141516171819202122232425262728 <GetResponse> <GetResponseWithDataBlock> <InvokeIdAndPriority Value="C1" /> <Result> <LastBlock Value="00" /> <BlockNumber Value="00000001" /> <Result> <RawData Value="09320102030405060708091011121314 1516171819202122232425262728" /> </Result> </GetResponseWithDataBlock> </GetResponse> // 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 <ReadResponse Qty="0001" > <DataBlockResult> <LastBlock Value="00" /> <BlockNumber Value="0001" /> <RawData Value="01000932010203040506070809101112 13141516171819202122232425262728 29" /> </DataBlockResult> </ReadResponse> // 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 SEQUENCE OF CHOICE.</pre>
<pre>C002C1 00000001 <GetRequest> <GetRequestForNextDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </GetRequestForNextDataBlock> </GetRequest></pre>	<pre>0501 050001 <ReadRequest Qty="0001" > <BlockNumberAccess> <BlockNumber Value="0001" /> </BlockNumberAccess> </ReadRequest></pre>
<pre>C402C1 01 00000002 00</pre>	<pre>0C01 02 01 0002</pre>

<pre> 16 29303132333435363738394041424344 454647484950 <GetResponse> <GetResponsewithDataBlock> <InvokeIdAndPriority Value="C1" /> <Result> <LastBlock Value="01" /> <BlockNumber Value="00000002" /> <Result> <RawData Value="29303132333435363738394041424344 454647484950" /> </Result> </Result> </GetResponsewithDataBlock> </GetResponse> // APDU length 32 bytes, 22 bytes of raw-data carries the // remaining part of data requested. </pre>	<pre> 15 30313233343536373839404142434445 4647484950 <ReadResponse Qty="0001" > <DataBlockResult> <LastBlock Value="01" /> <BlockNumber Value="0002" /> <RawData Value="30313233343536373839404142434445 4647484950" /> </DataBlockResult> </ReadResponse> // APDU length 28 bytes, 21 bytes of raw-data carries the // remaining part of data requested. </pre>
--	---

Table 128 – Example: Reading the value of a list of attributes with block transfer

<pre> 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> </pre>	<pre> 0502 020100 020110 <ReadRequest Qty="0002" > <VariableName Value="0100" /> <VariableName Value="0110" /> </ReadRequest> </pre>
--	---

DLMS User Association, DLMS/COSEM Architecture and Protocols, Edition 8.1

<pre>C402C1 00 00000001 00 1E 02000932010203040506070809101112 1314151617181920212223242526 <GetResponse> <GetResponsewithDataBlock> <InvokeIdAndPriority Value="C1" /> <Result> <LastBlock Value="00" /> <BlockNumber Value="00000001" /> <Result> <RawData Value="02000932010203040506070809101112 1314151617181920212223242526" /> </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.</pre>	<pre>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.</pre>
<pre>C002C1 00000001 <GetRequest> <GetRequestForNextDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </GetRequestForNextDataBlock> </GetRequest></pre>	<pre>0501 05 0001 <ReadRequest Qty="0001" > <BlockNumberAccess> <BlockNumber Value="0001" /> </BlockNumberAccess> </ReadRequest></pre>
<pre>C402C1 01 00000002 00 1E 27282930313233343536373839404142 4344454647484950000A03303030 <GetResponse> <GetResponsewithDataBlock> <InvokeIdAndPriority Value="C1" /> <Result> <LastBlock Value="01" /> <BlockNumber Value="00000002" /> <Result></pre>	<pre>0C01 02 01 0002 1B 30313233343536373839404142434445 4647484950000A03303030 <ReadResponse Qty="0001" > <DataBlockResult> <LastBlock Value="01" /> <BlockNumber Value="0002" /> <RawData Value="30313233343536373839404142434445 4647484950000A03303030" /> </DataBlockResult></pre>

<pre> <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. </pre>	<pre> </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 </pre>
--	---

Table 129 – Example: Writing the value of a single attribute without block transfer

<pre> C101C1 00010000800000FF0200 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> </pre>	<pre> 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> </pre>
<pre> C501C1 00 <SetResponse> <SetResponseNormal> <InvokeIdAndPriority Value="C1" /> <Result Value="Success" /> </SetResponseNormal> </SetResponse> </pre>	<pre> 0D01 00 <WriteResponse Qty="0001" > <Success /> </WriteResponse> </pre>

Table 130 – Example: Writing the value of a list of attributes without block transfer

<pre> 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> </pre>	<pre> 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> </pre>
<pre> C505C1 02 00 00 <SetResponse> </pre>	<pre> 0D02 00 00 <WriteResponse Qty="0002" > <Success /> </pre>

<pre> <SetResponseWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <_DataAccessResult Value="Success" /> <_DataAccessResult Value="Success" /> </Result> </SetResponseWithList> </SetResponse> </pre>	<pre> <Success /> </WriteResponse> </pre>
--	---

Table 131 – Example: Writing the value of a single attribute with block transfer

<pre> C102C1 00010000800000FF0200 00 00000001 15 09320102030405060708091011121314 1516171819 <SetRequest> <SetRequestWithFirstDataBlock> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> <DataBlock> <LastBlock Value="00" /> <BlockNumber Value="00000001" /> <RawData Value="09320102030405060708091011121314 1516171819" /> </DataBlock> </SetRequestWithFirstDataBlock> </SetRequest> // 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 <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="00" /> <BlockNumber Value="0001" /> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> <ListOfData Qty="0001" > <OctetString Value="01020100010932010203040506070809 101112131415161718192021222324" /> </ListOfData> </WriteRequest> // 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>
<pre> C502C1 00000001 <SetResponse> <SetResponseForDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </SetResponseForDataBlock> </SetResponse> </pre>	<pre> 0D01 02 0001 <WriteResponse Qty="0001" > <BlockNumber Value="0001" /> </WriteResponse> </pre>
<pre> C103C1 </pre>	<pre> 0601 </pre>

<pre> 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. </pre>	<pre> 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. </pre>
<pre> C503C10000000002 <SetResponse> <SetResponseForLastDataBlock> <InvokeIdAndPriority Value="C1" /> <Result Value="Success" /> <BlockNumber Value="00000002" /> </SetResponseForLastDataBlock> </SetResponse> </pre>	<pre> 0D0100 <WriteResponse Qty="0001" > <Success /> </WriteResponse> </pre>

Table 132 – Example: Writing the value of a list of attributes with block transfer

<pre> C105C1 02 00010000800000FF0200 00010000800100FF0200 00 00000001 0A 02093201020304050607 <SetRequest> <SetRequestWithListAndFirstDatablock> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptorList Qty="0002" > </pre>	<pre> 0601 07 00 0001 01 091F 02020100020110020932010203040506 070809101112131415161718192021 <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="00" /> </pre>
---	--

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	460/500
-----------------------	------------	------------------------	---------

DLMS User Association, DLMS/COSEM Architecture and Protocols, Edition 8.1

<pre> <_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> <DataBlock> <LastBlock Value="00" /> <BlockNumber Value="00000001" /> <RawData Value="02093201020304050607" /> </DataBlock> </SetRequestWithListAndWithFirstDatablock> </SetRequest> // 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> <BlockNumber Value="0001" /> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> <ListOfData Qty="0001" > <OctetString Value="02020100020110020932010203040506 070809101112131415161718192021" /> </ListOfData> </WriteRequest> // 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>
C502C1 00000001 <SetResponse> <SetResponseForDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </SetResponseForDataBlock> </SetResponse>	0D01 02 0001 <WriteResponse Qty="0001" > <BlockNumber Value="0001" /> </WriteResponse>
C103C1 00 00000002 1F 08091011121314151617181920212223 242526272829303132333435363738 <SetRequest> <SetRequestWithDataBlock> <InvokeIdAndPriority Value="C1" /> <DataBlock> <LastBlock Value="00" /> <BlockNumber Value="00000002" /> <RawData Value="08091011121314151617181920212223" /> </DataBlock> </SetRequestWithDataBlock> </SetRequest>	0601 07 00 0002 01 091F 22232425262728293031323334353637 383940414243444546474849500A03 <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="00" /> <BlockNumber Value="0002" /> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> </WriteRequest>

DLMS User Association, DLMS/COSEM Architecture and Protocols, Edition 8.1

<pre> </DataBlock> </SetRequestWithDataBlock> </SetRequest> // The APDU is 40 bytes. 31 bytes of raw-data contain the second // part of data to be written. </pre>	<pre> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> <ListOfData Qty="0001" > <OctetString Value="22232425262728293031323334353637 383940414243444546474849500A03" /> </ListOfData> </WriteRequest> // The APDU is 40 bytes. 31 bytes of octet-string contains raw- // data: the second 29 bytes of the first data to be written and // the data type and length of the second data to be written. The // value follows in the next block. </pre>
<p>C502C100000002</p> <pre> <SetResponse> <SetResponseForDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000002" /> </SetResponseForDataBlock> </SetResponse> </pre>	<pre> 0D01 02 0002 <WriteResponse Qty="0001" > <BlockNumber Value="0002" /> </WriteResponse> </pre>
<p>C103C1 01 00000003 11 3940414243444546474849500A033030 30</p> <pre> <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. </pre>	<pre> 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. </pre>
<p>C504C1 02 00 00 00000003</p> <pre> <SetResponse> </pre>	<pre> 0D02 00 00 <WriteResponse Qty="0002" > <Success /> <Success /> </WriteResponse> </pre>

DLMS User Association, DLMS/COSEM Architecture and Protocols, Edition 8.1

```
<SetResponseForLastDataBlockWithList>
  <InvokeIdAndPriority Value="C1" />
  <Result Qty="0002" >
    <_DataAccessResult Value="Success" />
    <_DataAccessResult Value="Success" />
  </Result>
  <BlockNumber Value="00000003" />
</SetResponseForLastDataBlockWithList>
</SetResponse>
```

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	463/500
-----------------------	------------	------------------------	---------

14.2 ACCESS service example

Table 133 shows an example of the ACCESS service without general block transfer.

Table 133 – Example: ACCESS service without block transfer

Message Elements (MAX APDU = 1024)	Contents	LEN (Bytes)
Access-Request	D9	1
long-invoke-id-and-priority	40000000	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		
SEQUENCE OF Data	04	1
null-data	00	1
null-data	00	1
array	01040203090100090CFFFFFFF 00000000000901FF0203090101090CF FFFFFFF00000000000901FF0203 090102090CFFFFFFF00000000 000901FF0203090103090CFFFFFFF FFFF00000000000901FF	90
array	0104020809010011FF11FF11FF11 FF11FF11FF0208090101110211011101 1101110111011101020809010211FF11 FF11FF11FF11FF11FF0208090103 1101110211021102110211021102	78

Complete Access-Request APDU (encoded)	D940000000000040100010000600100FF 02010008000010000FF020200140000 0D0000FF070200140000D0000FF0804 000001040203090100090CFFFFFFFFF FFFFF0000000000901FF020309010109 0CFFFFFFFFF000000000000901FF 0203090102090CFFFFFFFFF0000 0000000901FF0203090103090CFFFFF FFFFFFF0000000000901FF01040208 09010011FF11FF11FF11FF11FF11 FF02080901011021101110111011101 11011101020809010211FF11FF11FF11 FF11FF11FF11FF020809010311011102 11021102110211021102	218
Access-Response	DA	1
long-invoke-id-and-priority	40000000	4
date-time	00	1
access-response-body		0
access-request-specification OPTIONAL	00	1
access-response-list-of-data		0
SEQUENCE OF Data	04	1
octet-string	09083030303030303031	10
octet-string	090C07DC030C07161E0000FF8880	14
null-data	00	1
null-data	00	1
access-response-specification		0
SEQUENCE OF CHOICE	04	1
access-response-get	01	1
result	00	1
access-response-get	01	1
result	00	1
access-response-set	02	1
result	00	1
access-response-set	02	1
result	00	1
Complete Access-Response APDU (encoded)	DA4000000000000409083030303030 3031090C07DC030C07161E0000FF8880 0000040100010002000200	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.

The compact-array data type is present from the beginning in the DLMS/COSEM specification, but so far its interpretation was not unambiguous, therefore it has not been used.

The objective of this subclause 14.3 is to facilitate the use of compact-array encoding.

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
Total	17	13	4

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.

- 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
Total	44	40	4

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

Encoding as array (using the null-data feature)	Encoding as compact-array
Notice that 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. Notice also, that 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
Total	1361	982	375
Encoded data in % of raw data	68%	49%	

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%.

**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:

Table A. 1 – ECC_P256_Domain_Parameters

Parameter name	Symbol	Value
Field size	q	FFFFFFF 0000001 0000000 0000000 0000000 FFFFFFFF FFFFFFFF FFFFFFFF $= (2^{224} (2^{32}-1) + 2^{192} + 2^{96} - 1)$
Field representation indicator	FR	NULL
Curve parameter	a	FFFFFFF 0000001 0000000 0000000 0000000 FFFFFFF FFFFFFF FFFFFFC
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 0000000 FFFFFFF FFFFFFF BCE6FAAD A7179E84 F3B9CAC2 FC632551
Cofactor	h	1

Table A. 2 – ECC_P384_Domain_Parameters

Parameter name	Symbol	Value
Field size	q	FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFFF 00000000 00000000 FFFFFFFF $=2^{384}-2^{128}-2^{96}+2^{32}-1$
Field representation indicator	FR	NULL
Curve parameter	a	FFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE FFFFFFFFF 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
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	FFFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFFF FFFFFFFF C7634D81 F4372DDF 581A0DB2 48B0A77A ECEC196A CCC52973
Cofactor	h	1

Annex B
(informative)
Example of an End entity signature certificate
using P-256 signed with P-256

Version	3
Serial Number	71
Signature Algorithm	ecdsa-with-SHA256
Issuer	O=DLMS-PKI,CN=SUB-CA
Validity	Not Before: Jan 1 00:00:00 1970 GMT Not After : Dec 31 23:59:59 9999 GMT
Subject	CN=MM12345678
Public Key Algorithm	id-ecPublicKey
Public-Key	Pub:04:f5:44:80:11:79:bb:4e:30:86:95:2b:8d:e4:8e:ba:79:57: cf:19:ad:5b:d3:f7:ec:b1:31:bf:71:9a:1c:2f:e7:8a:93:48:d7:66: d0:67:c5:fb:c1:4b:25:8a:03:a4:6a:b0:f0:0a:09:9f:88:7e:6a:d2: 20:05:e6:03:9b:70:1e
ASN1 OID	prime256v1
Authority Key Identifier	keyid:9D:BA:19:85:19:73:DA:7E:C7:71:55:B2:30:EF:A1:BD:F5:DA:80:F9
Key Usage	Critical,Key Agreement
Signature Algorithm: ecdsa-with-SHA256	30:44:02:20:1b:87:dd:69:07:8a:73:22:7e:2f:43:ba:7c:b0: e5:13:9d:f2:aa:6b:f8:7c:ea:83:e2:fc:09:8f:e9:60:99:d6: 02:20:28:d7:0c:bc:cf:45:24:46:ab:e2:58:2e:a4:94:05:d9: 7b:2e:79:57:c9:3c:40:4f:d0:49:39:2b:e7:db:a0:63

Field	Value	Comments
<i>begin tbsCertificate</i>		
version	2	X.509 version 3
serialNumber	INTEGER	Positive, 20 octets or less
signature	1.2.840.10045.4.3.2	ECDSA with SHA-256
validity		Follows RFC5280
subject		Follows RFC5280, if empty, the subjectAltName must be present and Critical
Unique Identifiers		
subjectUniqueId	Bit string	Optional
<i>subjectPublicKeyInfo</i>		
AlgorithmIdentifier		
	algorithm	1.2.840.10045.2.1 EC
	parameters	1.2.840.10045.3.1.7 P-256 named curve
subjectPublicKey	bit string 528 bits	1 st byte = 0, 2 nd byte = 4 (uncompressed) 256 bit x, 256 bit y coordinates
<i>Extensions</i>		
Authority Key Identifier		
Identifier	2.5.29.35	Follows RFC5280
Value	Octet String	8 or 20 octets
Critical	False	
Key Usage		0 digitalSignature 4 keyAgreement 5 keyCertSign 6 cRLSign At least one bit must be 1
Identifier	2.3.29.15	
Value	DER encoded bit string	
critical	True	
subjectAltName		
Identifier	2.5.29.17	
Value	OID(s)	

Critical	True when CN is absent	
CertificatePolicies Identifier Value critical	2.5.29.32 OID Depends on companion spec.	
Subject Key Identifier Identifier Value Critical	2.5.29.14 Octet String False	Follows RFC5280, applicable to CAs 8 or 20 octets
<i>end tbsCertificate</i>		
signatureAlgorithm	1.2.840.10045.4.3.2	ECDSA with SHA-256
signatureValue	Bit string	Encoded bit string value of a DER encoded sequence of 2 integers; each a maximum of 33 bytes.

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.

Prerequisites: The initiator and the responder use the same elliptic curve and key derivation method

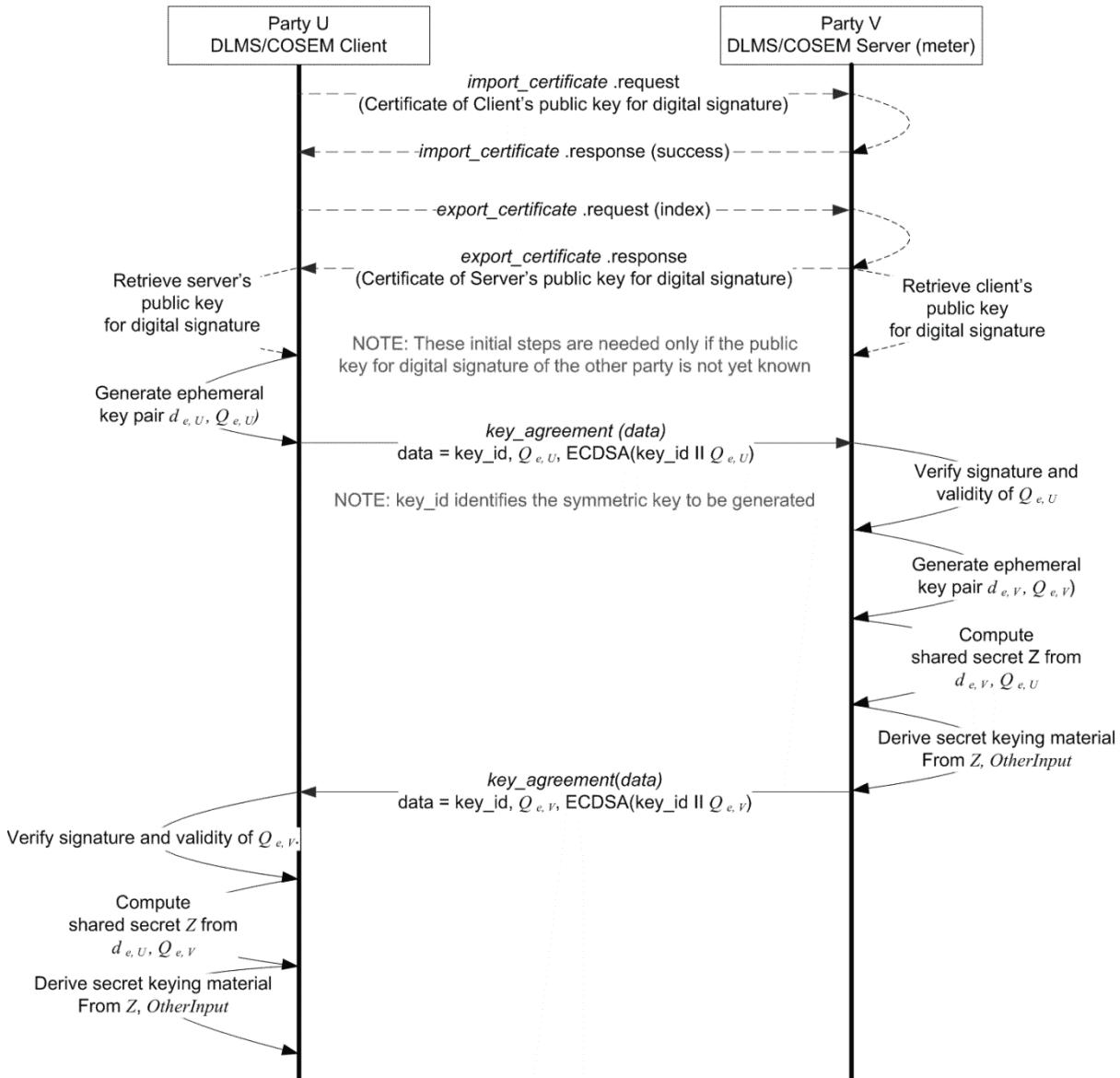


Figure C. 1 – MSC for key agreement using the Ephemeral Unified Model C(2e, 0s, ECC CDH) scheme

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 $(key_id, Q_{e,u})$ 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 $(key_id, Q_{e,v})$ 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 DD827E820BCA2BCC64362FB83D86A82B87BB8B7161D2 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	95F41066009B185B074F5FFF736B71C325FCADB2BC0C 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
Ephemeral Public Key Signature Server	Epri-K-Sig-S	A92995225CEE004ED4376057EEE9536E97EE6F5BAE43E 59BDBBD515A89FB2CB83F2A270871A31B09338DCF0D17 97466087908BA4A6ED8FD48B9EA067DA67DC4D // The key_id Epri-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 4E9A01CE9A0E55026BCAA4EEBEB764CED64BB34 // 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	60857406080300 // AES-GCM-128	7	56
KDF(Z,AlgID,Sys-TC,Sys-TS)	KDF	E025CA6F9EE8D2B40F993739D44CFBC092716422CEEA F60E8414ADD73F5BFDB7	32	256
Global Unicast Encryption Key	GUEK	E025CA6F9EE8D2B40F993739D44CFBC0	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

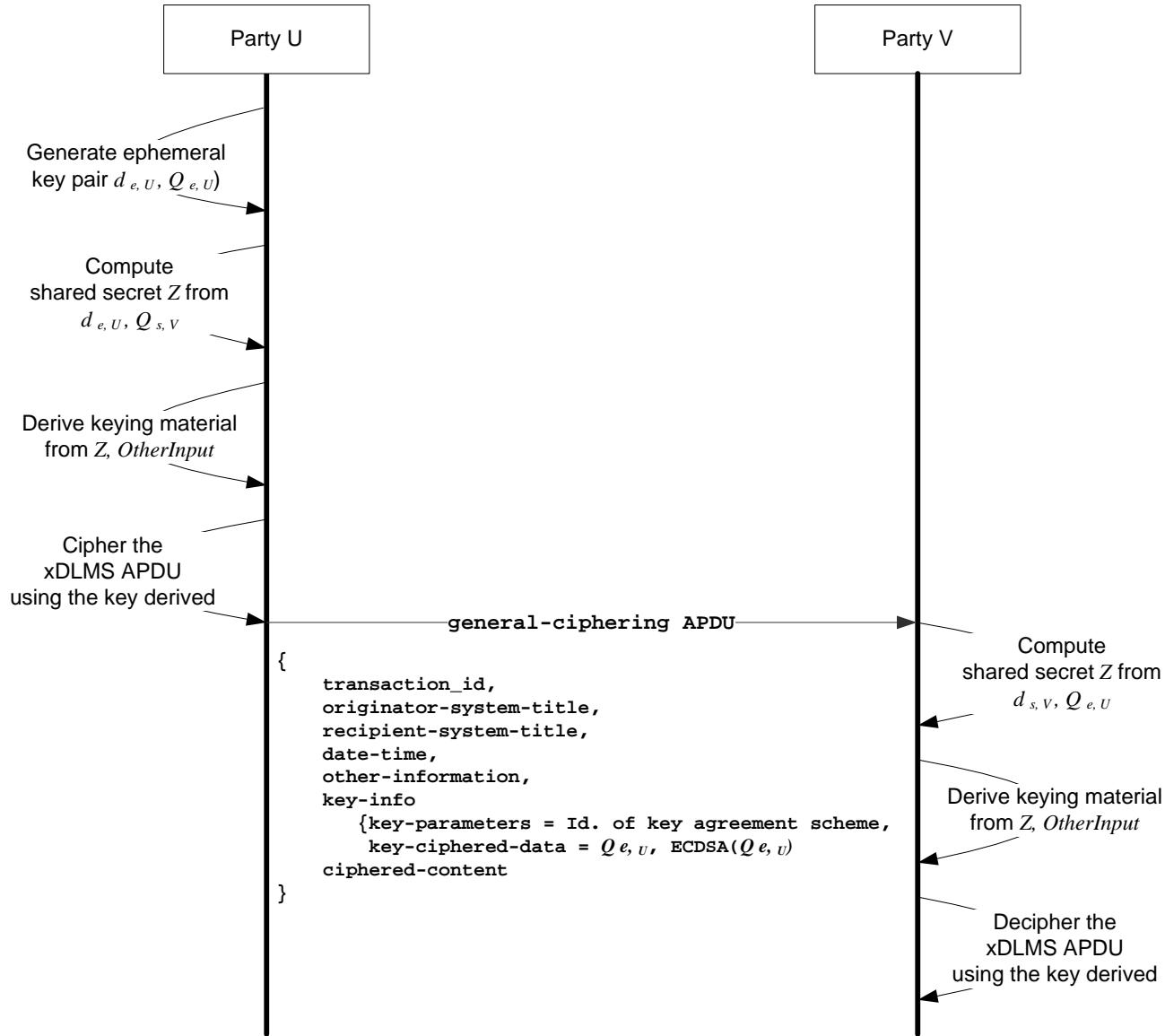


Figure C. 2 – CIPHERED xDLMS APDU PROTECTED BY AN EPHEMERAL KEY ESTABLISHED USING THE ONE-PASS DIFFIE-HELLMAN (1e, 1s, ECC CDH) SCHEME

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;
 - 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 21;
 - 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 ⁴ 5780289F761BB21 BFB0835CB5585E8B373	32	256
Private Signing Key Server	Pri-KS	AE55414FFE079F9FC95649536BD1C2B5653D200813727 E07D501A8B550C69207	32	256
Public Signing Key Client	Pub-KC	BAAFFDE06A8CB1C9DAE8D94023C601DBBB249254BA22E DD827E820BCA2BC64362FB83D86A82B87BB8B7161D2 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

Security material	Symbol	Contents	LEN(X) Bytes	len(X) Bits
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 54637616011B9C2378773295F0F850D8DAFD1BBE9FE66 6E53E4F097CD10B38B69622152724A90987444	64	512
Ephemeral Private Key Client	Epri-KC	47DAB03842E5B6E74828EF4F449B378D7DD1A5DAE1FFC A5AE0B0BE0AD18EC57E	32	256
Ephemeral Private Key Server	Epri-KS	819B1BEACC955E29139E368BF4119C126FF799EE16BCB A3F45C1EF16749BCB95	32	256
Ephemeral Public Key Signature Client	Epub-K-Sig-C	B51BE089D0B682863B2217201E73A1A9031968A9B4121 DCBC3281A69739AF87429F5B3AC5471E7B6A04A2C0F2F 8A25FD772A317DF97FC5463FEAC248EB8AB8BE // Epub-KC is included in signature	64	512
Ephemeral Public Key Signature Server	Epub-K-Sig-S	E1FF47974A1F6931A6502F58147463F0E8CC517D47F55 B0AC56D8AC5C9D0E481934F2D90F9893016BD82B6E3F FE21FF1588F3278B4E9D98EB4FB62ADD64B380 // 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 084D4D4D0000BC614E // key-parameters 8180C323C2BD45711DE4688637D919F92E9DB8FB2DFC2 13A88D21C9DC8DCBA917D8170511DE1BADB360D50058F 794B0960AE11FA28D392CFF907A62D13E3357B1DC0B51 BE089D0B682863B2217201E73A1A9031968A9B4121DCB C3281A69739AF87429F5B3AC5471E7B6A04A2C0F2F8A2 5FD772A317DF97FC5463FEAC248EB8AB8BE // key-ciphered-data 81EB3000000003F14FC102AE08241BC24EF12AA8049F 2C6C81B9248C47AD8DC4BB6BBE2DF0C96FDFA3722909D 156F2A0F02128C7CC404A0CE05898D077712D8997FAA3 90405C711FA771740D16290D73B11A2431843D9D4AE3A 3CD6B0EF5A811F9F979A90AE1937BFF27084C4169EA91 BE329D4893895AF46668E8C3938D40041F9A7859EB6F7 8FABF95FDD4465C5E0975B818FE1AA859514D299C1E0C DB7F3953FC5EE5598CEEBDA138CC9BD9E108CEE403A DC313C05510169F623147F4DDB2C5120B9FB77BD511DD 749A62CABAB3A4CA05F8136D33DC492C7C899649E07DF 7912638EDE1E69634A2B1A96 // ciphered-content	408	3264
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	233	1864

Security material	Symbol	Contents	LEN(X) Bytes	len(X) Bits
		<pre>084D4D4D00000000001 // key-parameters 81806439724714B47CD9CB988897D8424AB946DCD083D 37A954637616011B9C2378773295F0F850D8DAFD1BBE9 FE666E53E4F097CD10B38B69622152724A90987444E1F F47974A1F6931A6502F58147463F0E8CC517D47F55B0A C56DD8AC5C9D0E481934F2D90F9893016BD82B6E3FFE2 1FF1588F3278B4E9D98EB4FB62ADD64B380 // key-ciphered-data 3D30000000000D168C67323F1484B77BA50F35F8693A4 D9BFF236B71F10ABE857B6D22E1E235E30D50521552BB D97D0D99A72FADE4BB7C78573339706D7A // ciphered-content</pre>		
Shared Secret GC-C	Z-GC-C	0D4385BA0DD756CBCAB9887EB538396EE8F090A14C1079B43 59F115B977F4615	32	256
AlgorithmID GC-C	AlgID-GC-C	60857406080300 // AES-GCM-128	7	56
KDF(Z,AlgID,Sys-TC, Sys-TS) GC-C	KDF-GC-C	8CD08FF02EAC71712DE8449DC83318334962E71D78452 DBADF7BBFDBD76BB177	32	256
Encryption Key GC-C	EK-GC-C	8CD08FF02EAC71712DE8449DC8331833	16	128
Shared Secret GC-S	Z-GC-S	2B4302DC49790E2E78D990CFB52ED6E2F273DECE441A2 D95E4301B93812A9FAC	32	256
AlgorithmID GC-S	AlgID-GC-S	60857406080300	7	56
KDF(Z,AlgID,Sys-TS, Sys-TC) GC-S	KDF-GC-S	E357F06755CBF5C2C31457FE3CD1D5B83CF542A21B957 4591A5386AC2B260A48	32	256
Encryption Key GC-S	EK-GC-S	E357F06755CBF5C2C31457FE3CD1D5B8	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

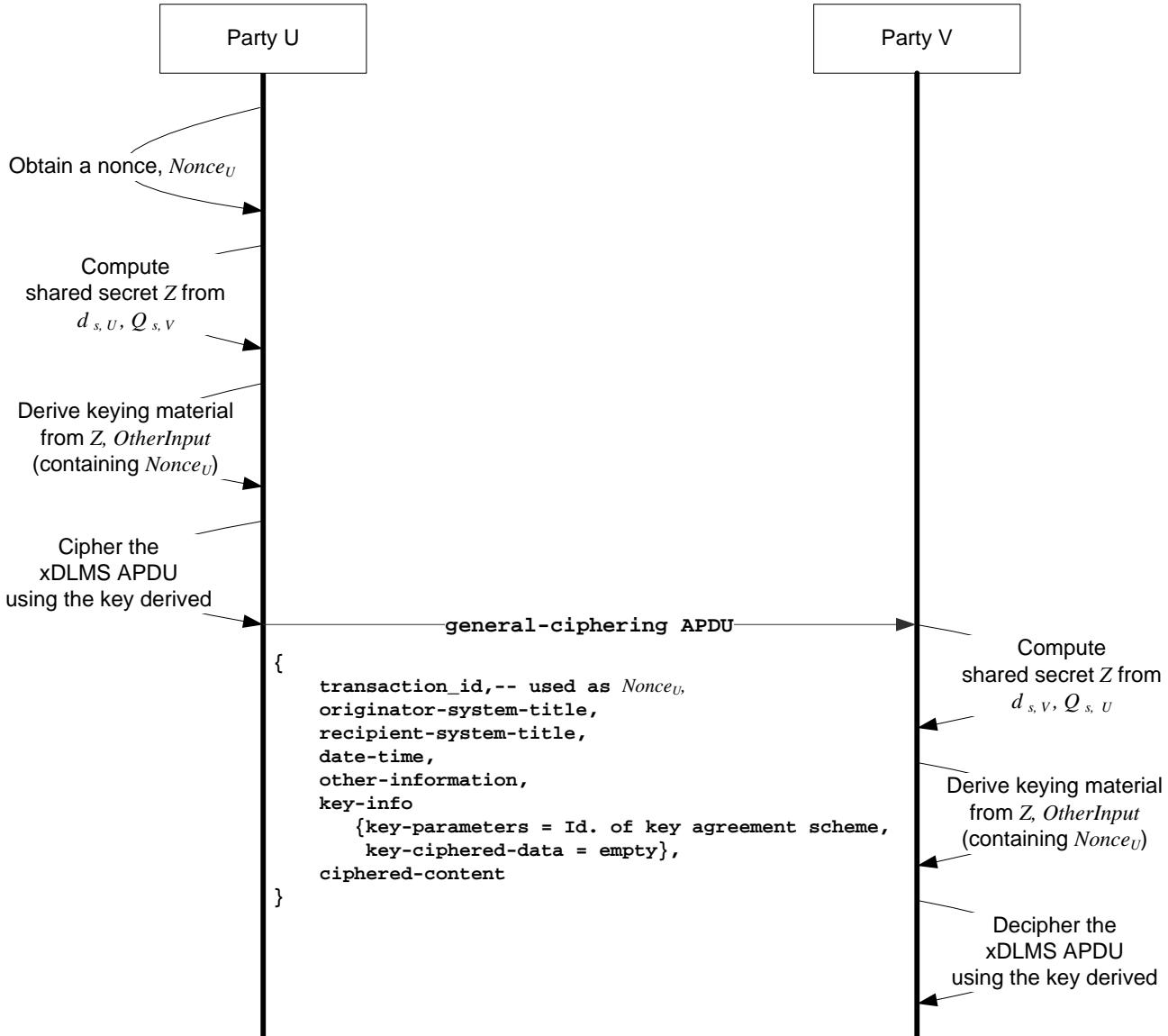


Figure C. 3 – Ciphered xDLMS APDU protected by an ephemeral key established using the Static Unified Model C(0e, 2s, ECC CDH) scheme

The process is the following:

- Step 1: The originator, taking the role of the Party U of the key agreement process obtains a nonce, $Nonce_U$;
- Step 2: It computes shared secret Z from $d_{s, U}, Q_{s, V}$;
NOTE See also Note to Table 16.
- Step 3: It derives the secret key from Z, and *OtherInput* that contains $Nonce_U$;

- 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: this field is used as $Nonce_U$;
 - 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 21.
 - 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 $Nonce_U$;
- Step 8: It deciphers the xDLMS APDU using the key derived.

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	483/500
-----------------------	------------	------------------------	---------

**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.

NOTE 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](#).

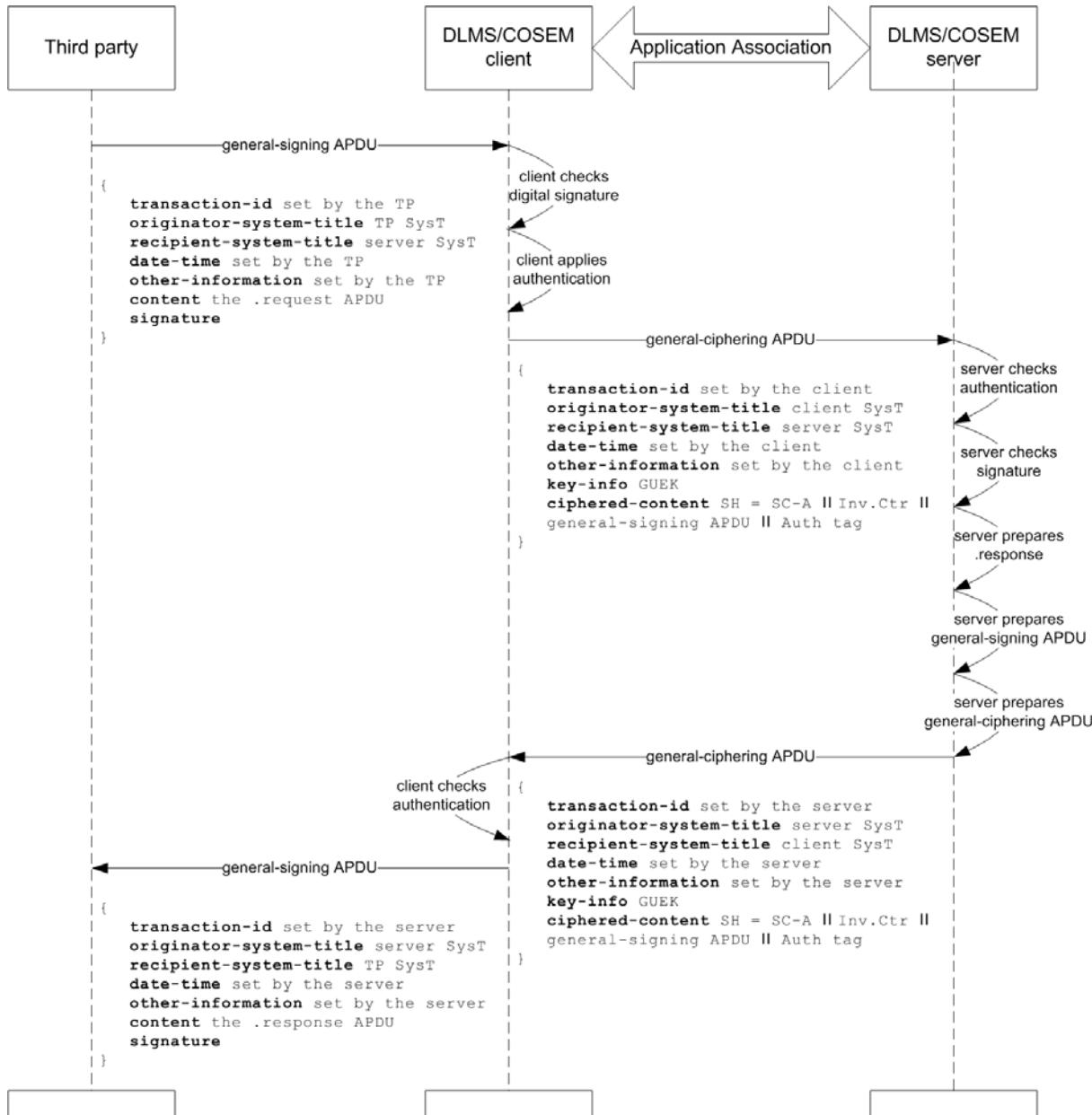


Figure D. 1 – Exchanging protected xDLMS APDUs between TP and server: example 1

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.

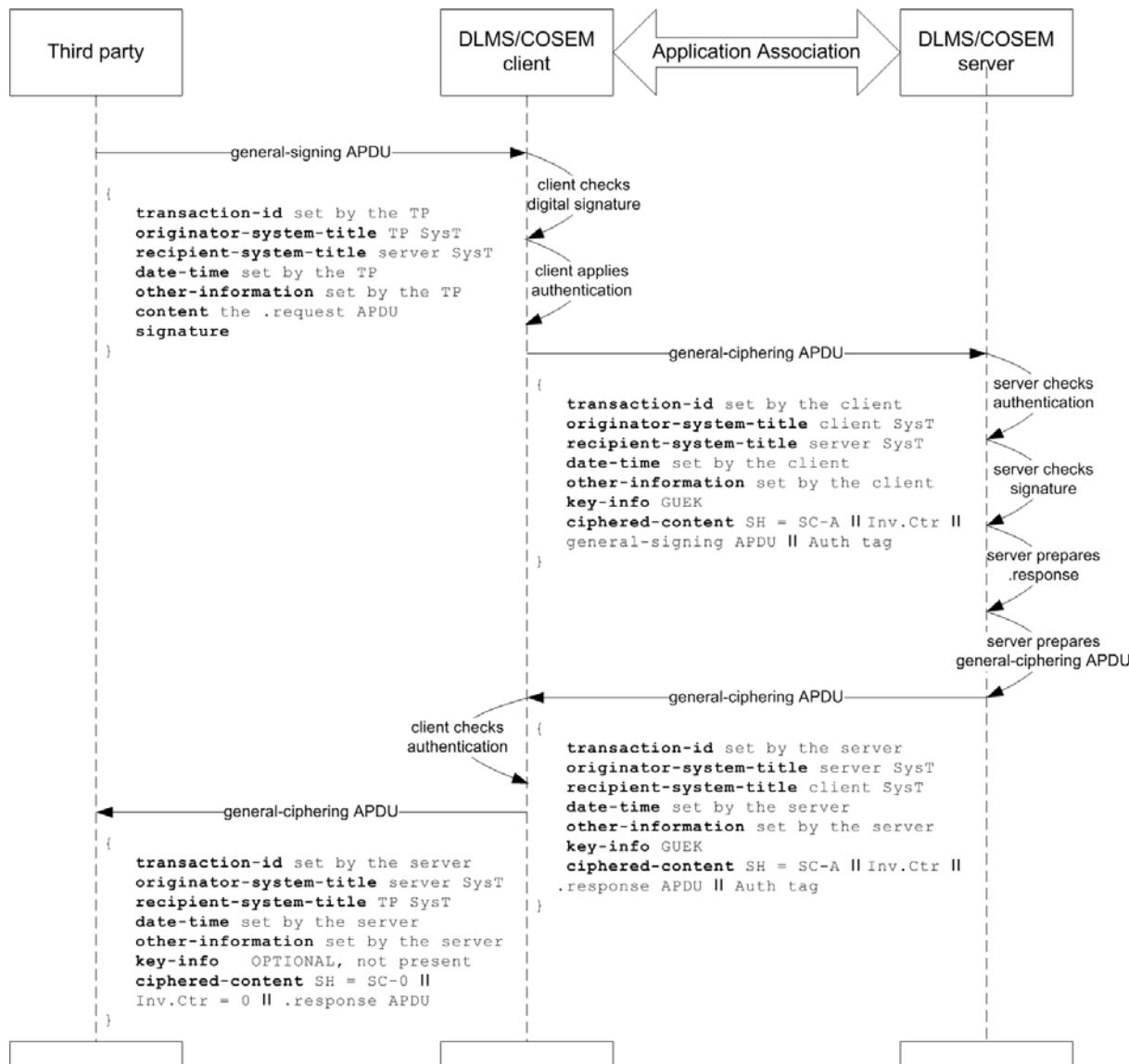


Figure D. 2 – Exchanging protected xDLMS APDUs between TP and server: example 2

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 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 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 62056-7-6:2013 *ELECTRICITY METERING DATA EXCHANGE - The DLMS/COSEM suite - Part 7-6: The 3-layer, connection-oriented HDLC based communication profile*

IEC 62056-8-3:2013 *ELECTRICITY METERING DATA EXCHANGE - The DLMS/COSEM suite - Part 8-3: PLC S-FSK communication profile for neighbourhood networks*

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	487/500
-----------------------	------------	------------------------	---------

IEC 62056-9-7:2013 *ELECTRICITY METERING DATA EXCHANGE – The DLMS/COSEM suite – Part 9-7: Communication profile for TCP-UDP/IP networks*

CLC/52056-8-4:2015, *ELECTRICITY METERING DATA EXCHANGE – THE DLMS/COSEM SUITE – Part 8-4: The narrowband OFDM PLC profile for PRIME networks*

CLC/TS 52056-8-5:2015, *ELECTRICITY METERING DATA EXCHANGE – THE DLMS/COSEM SUITE – Part 8-5: The narrowband OFDM PLC profile for G3-PLC 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 9798-2 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.211:1995, *Information technology – Open Systems Interconnection – Physical service definition*

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*

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: *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, John Viega, Secure Software, 4100 Lafayette Center Drive, Suite 100, Chantilly, VA 20151, 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>

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	488/500
-----------------------	------------	------------------------	---------

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 822 *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 4211 *Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF)*, 2005, <http://www.ietf.org/rfc/rfc4211>

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	489/500
-----------------------	------------	------------------------	---------

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 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>

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:

https://www.bsi.bund.de/DE/Themen/SmartMeter/TechnRichtlinie/TR_node.html

Index

- 16-bit FCS computation method 146
 3-layer, connection-oriented, HDLC based communication profile 347
 AA, confirmed 153, 157
 AA, pre-established 153, 157
 AA, unconfirmed 153, 157
 AARE APDU, BER encoding 427, 437
 AARQ and AARE encoding examples 418
 AARQ APDU, BER encoding 424, 434
 A-ASSOCIATE service 153, 223, 265
 Abort sequence 133
 Abstract syntax 379
 Abstract Syntax Notation 1 324
 Abstract syntax, COSEM APDUs 28, 310
 Access right 32, 58, 59, 162, 165, 205
 ACCESS service 155, 245, 288, 464
 Access_Request_Action 249
 Access_Request_Body 248
 Access_Request_Get 249
 Access_Request_List_Of_Data 249
 Access_Request_Set 249
 Access_Request_Specification 248, 249
 Access_Response_Body 249
 Access_Response_List_Of_Data 249
 Access_Selection_Parameters 237, 240
 ACSE functional units 28, 265
 ACSE procedures 161
 ACSE protocol version 225, 274
 ACSE requirements 267
ACSE services and APDUs 265
 ACSE, connection oriented 153
 ACSE_Requirements 225
 ACTION service 155, 241, 286
 ACTION.confirm 245
 ACTION.indication 245
 ACTION.request 244
 ACTION.response 245
 Action-Request 245
 ACTION-REQUEST-FIRST-BLOCK243, 287, 291, 295
 ACTION-REQUEST-LAST-BLOCK243, 287, 292, 295
 ACTION-REQUEST-NEXT 243, 287, 291
 Action-Request-Next-Pblock 287
 Action-Request-Normal 287
 ACTION-REQUEST-NORMAL243, 287, 291, 295, 298
 ACTION-REQUEST-ONE-BLOCK243, 287, 291, 295
 Action-Request-With-First-Pblock 287
 Action-Request-With-List 287
 ACTION-REQUEST-WITH-LIST243, 287, 292, 296, 298
 ACTION-REQUEST-WITH-LIST-AND-FIRST-BLOCK 243, 287, 292, 296
 Action-Request-With-List-And-With-Frist-Pblock 287
 Action-Request-With-Pblock 287
 Action-Response 245
 ACTION-RESPONSE-LAST-BLOCK243, 287, 292
 ACTION-RESPONSE-NEXT243, 287, 292, 296
 Action-Response-Next-Pblock 287
 Action-Response-Normal 287
 ACTION-RESPONSE-NORMAL243, 287, 292, 296
 ACTION-RESPONSE-ONE-BLOCK 243, 287
 ACTION-RESPONSE-ONE-ONE-BLOCK 292
 Action-Response-With-List 287
 ACTION-RESPONSE-WITH-LIST243, 287, 292, 296
 Action-Response-With-Pblock 287
 Active HDLC channel state 133
 Active initiator 362, 369
active OPEN 104
 Additional Authenticated Data 172, 174, 186, 209
 Additional data 172
 Address lengths, inopportune 129
Addressing 55, 356
 Addressing capability (wPort) 86
 Addressing, HDLC 127
 Addressing, M-Bus 396
 Addressing, S-FSK 379
 Advanced Encryption Standard 170, 171, 185
AES key wrap 174
 AES-GCM algorithm 205, 207
 Agreed_Key_Options 235
 agreed-key 188
 AL, management services 261
 ALARM state 381
 Alarm_Descriptor 372
 AlgorithmID 183
 All Physical 379
 All-configured 379
 All-L-SAP 379
 All-station (Broadcast) address 128
 All-station address 129
 Always repeater 371
 Application association 53, 57, 153, 228
 Application association, confirmed 58, 271, 359
 Application association, confirmed AA 350
 Application association, establishment 271, 381, 385
 Application association, graceful release 275
 Application association, non-graceful release 278
 Application association, pre-established 57, 275
 Application association, release 275, 383
 Application association, unconfirmed 58, 275, 350, 359
 Application context 58
 Application context name 153, 225, 268, 273
 Application entity 53
 Application layer 366

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	491/500
-----------------------	------------	------------------------	---------

Application layer message security.....	165
Application process	53, 60, 152, 366
Application process identification	100
Application Programming Interface.....	157
Application Service Object	152
Application_Addresses parameter	252
application-context-name	267
A-RELEASE	228
A-RELEASE service	153, 265, 360
ASN.1	269
Association Control Service Element.	152, 153
Association LN	53
Association SN.....	53
Asymmetric key algorithm	168
Attribute	31
Attribute_0 referencing.....	156
Authenticated decryption.....	172
Authenticated encryption.....	171, 172
Authentication	32, 58, 161, 162, 165, 168, 169, 175, 185, 207
Authentication functional unit	265
Authentication key.....	174, 186, 209, 214
Authentication key, GAK.....	186, 188, 189
Authentication mechanism.....	58, 162
Authentication mechanism name	153, 269
Authentication tag	172, 173, 174
Authentication, High Level Security	162, 164, 226, 274
Authentication, Low Level Security	162, 164, 226
Authentication, Lowest Level Security	162, 226
Authentication, no security	162, 163
Authenticity	169
Automatic configuration, Repeater Call service	369
A-XDR encoding.....	74
Battery operated devices.....	403
Binding.....	55
Bit String	176
Block cipher	170
Block cipher algorithm	170
Block cipher key	173, 174, 186
Block transfer, bi-directional	159
Block transfer , general	159, 280, 286
Block transfer, service-specific... <td>157, 280, 286</td>	157, 280, 286
Block_Number	237, 241, 244, 256, 259, 282, 285
Block_Number_Access.....	253, 255, 256
Block_Transfer_Streaming	302
Block_Transfer_Window	302
block-data	302
block-number	302
block-number-acknowledged	302
Broadcast.....	57
Broadcasting	384, 386
Broker	60, 167, 191
Busy.....	144
Called_AE_Invocation_Identifier.....	225
Called_AE_Qualifier.....	225
Called_AP_Invocation	225
Called_AP_Title	225
Calling authentication value.....	226
CALLING Physical Device	129
CALLING Physical Device Address....	128, 354
calling_AE_invocation_identifier:	273
Calling_AE_Qualifier	225
Calling_AP_Invocation_Identifier	225
calling-AE-qualifier	273
calling-authentication-value	267
Certificate and certificate extension profile	190, 193
Certificate extension	197
Certificate extension, Authority Key Identifier	198
Certificate extension, Basic constraints	199
Certificate extension, CertificatePolicies ...	198
Certificate extension, cRLDistributionPoints	200
Certificate extension, Extended Key Usage	200
Certificate extension, IssuerAltName	199
Certificate extension, KeyUsage	198
Certificate extension, SubjectAltNames	198
Certificate extension, SubjectKeyIdentifier	198
Certificate Policy	192, 193
Certificate removal	204
Certificate Signing Request	191
Certificate, client	203
Certificate, digital signature key	193
Certificate, Issuer	195
Certificate, Serial number	195
Certificate, static key agreement key	193
Certificate, Subject	195
Certificate, Subject Unique ID	197
Certificate, SubjectPublicKeyInfo	196
Certificate, Validity period	196
Certificate, X.509 v3	193
Certificates, server	203
Certification Authority	191
Challenge, <i>CtoS</i>	164
Challenge, <i>StoC</i>	164
<i>change_HLS_secret</i>	164
Check Initiator Phase	373, 375, 378
CIASE	380
CIASE L-SAP	379
CIASE protocol.....	56
CI-PDU.....	440
Ciphered APDU	32
Ciphered xDLMS APDUs	206
Ciphertext	169, 173, 185
Clear Alarm, S-FSK	450
ClearAlarm service	367, 371
Client	53, 54, 56, 57, 59, 60, 217
Client Management Process	56, 91, 128, 380, 401, 413
Client side layer management services.....	261
Client SN Mapper ASE	259, 261
Client SN_Mapper ASE	152, 257
Client system title	273
Client user	225
Client user identification	57, 162, 273

Client/server model	54
Client_LLC_Address.....	349
Client_MAC_Address	349
Client_Max_Receive_PDU_Size.....	226
client_system_title.....	57
client-max-receive-pdu-size.....	273
Collision	354
Command/response frame rejection	144
Common Name.....	195
Communication environment	346
Communication profile.....	60
Communication profile specific parameters	347
Communication profile structure	346
Compact array.....	466
Composable xDLMS messages.....	158
Compression.....	53, 153, 158, 184, 207
Confidentiality	168, 169
Configuration Initiation Application Service Element	366
Confirmed service	154
Confirmed service invocations.....	279
confirmedServiceError.....	256, 259, 279
Conformance block	159, 246, 278, 419
Conformance testing	65
Connection managers	60
Connection oriented	57
Context negotiation	153
Control field	126
Control function.....	152, 263
Convergence layer	60
Copyright	25
COSEM APDU, abstract syntax.....	310
COSEM application context.....	157, 159
COSEM application context name	268, 269
COSEM authentication mechanism	58
COSEM authentication mechanism name.....	269, 270
COSEM Cryptographic algorithm ID-s	270
COSEM data protection.....	32, 220
COSEM data security.....	168
COSEM Glossary of Terms	32
COSEM logical device address	349
COSEM object32, 48, 53, 58, 59, 60, 61, 64, 154, 155, 156, 162, 163, 165, 167, 168, 205, 206, 220, 223, 236, 237, 238, 240, 243, 245, 246, 249, 250, 252, 254, 255, 257, 258, 259, 260, 261, 279, 280, 283, 288, 290, 293, 294, 298	
COSEM object model	53
COSEM physical device address	349
COSEM_Attribute_Descriptor	237, 240, 249
COSEM_Authentication_Mechanism_Name	274
COSEM_Class_Id	237, 240, 243
COSEM_Method_Descriptor.....	243
COSEM_Method_Id.....	243
COSEM_Object_Attribute_Id	237, 240
COSEM_Object_Instance_Id	237, 240, 243
COSEM_on_IP.....	28, 61, 355
COSEM-ABORT	351
COSEM-ABORT service	153, 230
COSEM-OPEN service	153, 223, 271, 350, 360
COSEM-OPEN service invocations, repeated	274
COSEM-RELEASE service	153, 228, 275, 350
Counter mode.....	171
Country	195
Critical certificate extension.....	193
Cross-talk.....	373
Cryptographic algorithm	168
Cryptographic algorithm IDs	270
Cryptographic protection	153
Cryptoperiod	189
Data	220
Data collection system	62
Data Communication Equipment.....	66
Data conversions.....	176
Data integrity	168, 175
Data length.....	86, 90
Data link connection, disconnecting.....	116
Data link connection, setting up.....	112
Data link layer, HDLC	84, 110, 347
Data link layer, management services	149
Data link layer, S-FSK	365
Data link, disconnection	136
Data Terminal Equipment	66
Data transfer	120
Data transfer services, protocol	278
Data_Access_Error	256, 259
Data_Access_Result	241
Data_Length.....	88
DataBlock_G	237, 282
DataBlock_SA	240, 244, 285
DataNotification service	59, 154, 155, 251, 289, 403
Date_Time.....	235
date-time	213
Decryption.....	169
Dedicated key	186, 226
Denial-of-service attack	228, 383
Destination address	126, 129
Destination wPort	90
Deterministic construction	173
Device identification number	398
Device type	398
Device version	398
Digital signature ...	32, 165, 175, 177, 185, 217
Digital signature key pair	190
Direct local connection	81
Direct local data exchange	81
Directly trusted key.....	191
DISC command	136
DISC frame	118, 131
Discover service	367
DiscoverReport	367
Discovery and Registration process, S-FSK	376
DL-CONNECT service	112
DL-CONNECT service,	366
DL-CONNECT.confirm.....	115

DL-CONNECT.indication	114
DL-CONNECT.request	113
DL-CONNECT.response	114
DL-DATA service.....	120
DL-Data services.....	365
DL-DATA services, connectionless.....	366
DL-DATA services, connection-oriented	366
DL-DATA.confirm	123
DL-DATA.indication.....	122
DL-DATA.request	121
DL-DISCONNECT service	116
DL-DISCONNECT.confirm.....	119
DL-DISCONNECT.indication	118
DL-DISCONNECT.request.....	117
DL-DISCONNECT.response	119
DL-GET_VALUE.confirm	150
DL-GET_VALUE.request	150
DL-INITIALIZE.confirm	149
DL-INITIALIZE.request.....	149
DL-LM_EVENT.indication	151
DLMS conformance.....	226
DLMS named variable	156
DLMS version number	226
DLMS/COSEM AL	60, 62, 347
DLMS/COSEM AL, layer management services	160
DLMS/COSEM AL, protocol specification ..	161
DLMS/COSEM AL, service specification....	221
DLMS/COSEM AL, structure	152
DLMS/COSEM application layer	85, 152, 263, 346, 355
DLMS/COSEM aware	60
DLMS/COSEM client	63, 167
DLMS/COSEM client model.....	63
DLMS/COSEM conformance test process....	32
DLMS/COSEM security concept	162
DLMS/COSEM server model	62
DLMS/COSEM transport layer	27, 60, 62, 85, 355
DLMS/COSEM transport layer, state machine	91
DLMS/COSEM transport layer, TCP based..	92
DLMS/COSEM transport layer, TCP based, protocol specification	100
DLMS/COSEM transport layer, TCP based, service specification.....	92
DLMS/COSEM transport layer, UDP based .	87
DLMS/COSEM transport layer, UDP based, protocol specification	89
DLMS/COSEM transport layer, UDP based, service specification.....	87
DL-Reply services	365
DL-SET_VALUE.confirm.....	151
DL-SET_VALUE.request	150
DL-Update-Reply services.....	365
DM frame	132
DM response.....	132, 134, 136, 144
Domain parameters	175, 179, 190
Domain parameters, validity	190
Dynamic repeater	371
ECC_P256_Domain_Parameters	471
ECC_P384_Domain_Parameters	472
ECDSA algorithm	205
ECPoint.....	197
Elliptic curve	176, 178
Elliptic curve cryptography	175
Elliptic curve digital signature (ECDSA)	178
Encoding, A-XDR	270
Encoding, BER	270
Encryption	165, 169, 185, 207
Encryption key.....	173, 186
Encryption, Mode of Operation	170
End entity	193
End entity certificate	200
End entity signature certificate	473
End-to-end security	32, 60, 162
Ephemeral key	186
Ephemeral key agreement key pair.....	190
Ephemeral key pair	180
Ephemeral private key.....	179
Ephemeral public key	179
Ephemeral Unified Model	179, 189, 475
Error detection	92
Ethernet	63, 489
EventNotification service	59, 154, 155, 251, 290, 384, 386, 403
EventNotification service, 3-layer, CO, HDLC based profile	351
EventNotification service, TCP-UDP/IP based profile.....	361
exception-response	382
ExceptionResponse	280
Failure management	246
Failure_type	225
Fast FCS Implementation	146
Fast Synchronization	373, 375
FCS calculation	146
FCS error	132, 144
FCS table generator	147
Field Element	177
fin segment	102
Fixed field	173
Flag field	126
Flow control.....	92
FORGOTTEN	378
Format type sub-field.....	126
Frame Check Sequence (FCS)	127
Frame format field	126
Frame length sub-field.....	126
Frame reject response (FRMR)	132
FRMR response	132, 144
FTP	63
Full-duplex	92, 110
Galois/Counter Mode.....	170, 171, 185
Gateway protocol	414, 415
General block transfer	28, 157, 159, 230, 289, 299
General block transfer, abort	309
General protection.....	158
General_Block_Transfer_Parameters	299

general-ciphering	158, 212, 214
general-ded-ciphering	158, 211
general-glo-ciphering	158, 211
GeneralizedTime	196
general-signing	158
generate_certificate_request method.....	202
generate_key_pair method	202
Generic communication profile	61
GET service	155, 236, 280
GET.confirm.....	238
GET.indication	238
GET.request.....	238
GET.response	238
Get_Data_Result.....	237
Get-Request.....	238
Get-Request-Next	280
GET-REQUEST-NEXT	237, 280, 282, 291
Get-Request-Normal	280
GET-REQUEST-NORMAL.....	237, 280, 291
Get-Request-With-List.....	280
GET-REQUEST-WITH-LIST	237, 280, 291
Get-Response	238
GET-RESPONSE-LAST-BLOCK.....	237, 280, 291
Get-Response-Normal.....	280
GET-RESPONSE-NORMAL.....	237, 280, 291
GET-RESPONSE-ONE-BLOCK	237, 280, 282, 291
Get-Response-With-Datablock	280
Get-Response-With-List	280
GET-RESPONSE-WITH-LIST	237, 280, 291
Global broadcast encryption key, GBEK ...	186, 188, 189
Global key	186
Global unicast encryption key, GUEK186, 188, 189	
Graceful release.....	153
Half-duplex.....	66, 110
Hash function	168
Hash value	168
Hayes commands	76
HCS error.....	144
HDLC address	122
HDLC address field structure	127
HDLC addresses, special and reserved	128
HDLC based data link layer	62, 365
HDLC channel operation	134
HDLC channel states	133
HDLC extended addressing	127
HDLC frame	126
HDLC frame format type 3.....	110, 126
HDLC non-operational modes	134
HDLC operational modes	134
HDLC parameter negotiation	135
HDLC protocol.....	366
HDLC, CALLING device physical address .	143
HDLC, command and response frames	130
HDLC, elements of the procedures.....	132
HDLC, exception recovery	144
HDLC, exchange of information frames	137
HDLC, exchanging data	137
HDLC, Multi- and broadcasting	139
HDLC, Response time-out	144
HDLC, segmentation	138
HDLC, selected repertoire	130
HDLC, sequence number.....	137
HDLC, transferring long MSDUs from the server to the client	138
HDLC, Window size considerations	138
Head End System.....	363, 414
Header Check Sequence (HCS)	127
High Level Security authentication.....	162
HLS authentication	218
HLS authentication mechanism 3, MD5	218
HLS authentication mechanism 4, SHA-1 ..	218
HLS authentication mechanism 5, GMAC ..	219
HLS authentication mechanism 6, SHA-256	218
HLS authentication mechanism 7, ECDSA	218, 219
HLS secret	164
HTTP	63
I frame, command and response	130
I_COMPLETE	122, 137
I_FIRST_FRAGMENT	122, 137, 138
I_FRAGMENT	122, 137, 139
I_LAST_FRAGMENT	122, 137, 139
Identification	32, 162, 356
Identification and addressing scheme	346, 347
Identification protocol specification	74
Identification service.....	73, 352
Identified_Key_Options	235
identified-key	188
IDENTIFY.request	73
IDENTIFY.response	73
Identifying service invocations	157
Idle HDLC channel state	134
IDLE sub-state	104
Implementation_Information	226
implementation-information	267
import_certificate method	201, 202
Inactivity time-out	145
Information field	127
Information security	25, 27, 161
InformationReport service	59, 155, 261, 299, 403
InformationReport.request	299
informationReportRequest	299
Initial credit	369
Initialization vector	165, 170, 172, 173, 209
InitiateRequest APDU	420, 432
InitiateResponse APDU	421, 435
Initiator	101, 362, 379
Initiator L-SAP	379
Integer	176
Integrity	169
Intelligent Search Initiator process	367, 373
Interconnectivity	64
Interface class	
Communication media setup	60
interface model.....	31

Inter-frame time-out.....	145
Internet Protocol.....	85, 355, 356
Inter-octet time-out.....	134
Interoperability	64
Invalid Frame	133
Invocation counter.....	213
Invocation field.....	173
Invoke_Id	157, 236, 239, 242, 282, 286, 287
Invoke-Id, long	157
Invoke-Id-And-Priority	360
IP network	85
IPv4 address	357
Kernel functional unit	265
Key agreement.....	32, 175, 179, 185, 189
Key agreement, Ephemeral Unified Model	179
Key agreement, One-Pass Diffie-Hellman	180
Key agreement, Static Unified Model	181
Key derivation	179
Key Derivation Function	183
Key Encrypting Key, KEK ...	171, 174, 186, 189
Key identification	188
Key information	187
Key pair generation	190
Key size	185
Key transport	32
Key wrapping	171, 185, 188
Key wrapping key	171
<i>key_agreement</i> method	189
Key_Info_Options	235
Key_set subfield.....	208
<i>key_transfer</i> method.....	189
key-ciphered-data	188
key-info	213
key-parameters	188
Last_Block ..	237, 241, 244, 256, 258, 282, 285
last-block	303
Link Layer Address, M-Bus.....	396
Link Layer Address, wired M-Bus	397
Link Layer Address, wireless M-Bus	398
LLC addresses	379
LLC PDU format.....	124
LLC sublayer.....	110, 365
LLC sublayer, connectionless.....	365
LLC sublayer, HDLC based	365
LLC sublayer, protocol specification	124
LLC sublayer, state transition table	125
LN/SN data transfer service mapping	262
Local Network	53, 355
Local Network Access Points	363
Local_IP_Address	88
Local_or_Remote	225, 229
Local_TCP_Port	93
Local_UDP_Port.....	88
Local_wPort	88
Logical device	55, 61, 366
Logical Device Name	57
Logical Link Control	365
Logical name.....	156
Logical Name referencing.....	58, 154, 156
Long messages	384, 386
Long service parameters	157
Long_Invoke_Id	246, 247, 251
Long_M-Bus_Data_Header.....	394
Lost block recovery	159
Low Level Security authentication	162
Lower HDLC address	127
MAC address, HDLC	127
MAC address, S-FSK	379
MAC PDU.....	126
MAC sublayer	110, 365
MAC sublayer, data communication.....	124
MAC sublayer, physical layer services used by	123
MAC sublayer, protocol specification	126
MAC sublayer, state transition diagram	145
MA-CREATE service	112
MA-CREATE.confirm	115
MA-CREATE.indication.....	114
MA-CREATE.request	113
MA-CREATE.response	114
MA-DATA service	120
MA-Data services	365
MA-DATA.confirm.....	123
MA-DATA.indication	122
MA-DATA.request	121
MA-DISCONNECT service	116
MA-DISCONNECT.confirm	119
MA-DISCONNECT.indication	118
MA-DISCONNECT.request	117
MA-DISCONNECT.response	119
Management Logical Device	56, 62, 91, 252, 299, 380, 401, 413
Management Logical Device address	128
Manufacturer identification	398
Master key	186, 188, 189
Master/ Slave operation on the multi-drop bus	353
MA-Sync.indication.....	365
Max_Adr_MAC	370
MAX_NB_OF_RETRIES	135, 137, 145
Maximum information field length	145
MAXIMUM_INFORMATION_FIELD_LENGTH	135
M-Bus based transport layer.....	391
M-Bus broadcast	398
M-Bus Data Header, long	400
M-Bus data header, short	400
M-Bus link layer.....	391
M-Bus physical layer	391
M-Bus Physical layer	391
M-Bus profile , wired	389, 397
M-Bus profile , wireless	389
M-Bus TPDU format	399
M-Bus wrapper	396, 400
M-Bus, Client and server SAPs	401
M-Bus_Data_Header_Type	394
mechanism-name	267
Medium Access Control.....	365
Message Authentication Code	170
Message digest	168

Message integrity	171
Message security, client - server	166
Message security, end-to-end	167
Messaging.....	31
Messaging patterns	59
Meter installation.....	65
Metering equipment.....	61
METERING HDLC protocol.....	81
Method.....	31
mHLS authenticatin meccanism 5, GMAC)	218
Modelling	31
Multi- and broadcasting using UDP	88
Multi-drop configuration, 3-layer, CO, HDLC based profile	353
Multi-layer protection.....	32, 158, 165, 217
Multiple references.....	156
Mutual authentication	162
N(S) sequence error.....	144
Naming	55
Nb_Tslot_For_New.....	370
Negotiated_DLMS_Version_Number	226
Negotiated_xDLMS_Context	226
Neighbourhood Network	53, 355
Neighbourhood Network Access Point.....	363
Never repeater	371
NEW	379
NEW and LOCKED state	375, 378
NEW and UNLOCKED state	374, 376, 377, 378
New system	362
New system title	362
NIST Concatenation KDF	179
No TCP Connection.....	103
NO-BODY	377, 379
Non-basic frame format transparency	110
Nonce	173, 181
Non-critical certificate extension	193
None_M-Bus_Data_Header	394
Non-repudiation	168, 175
No-station address	128, 129
Notification_Body	251
NRM.....	133
NRM, Normal Response Mode	137
NSA Suite B	184, 471
OBJECT IDENTIFIER	196
Octet String	176
of Public Key Infrastructure	191
One-Pass Diffie-Hellman	179, 189, 214, 478
Order of bit and octet transmission.....	133
Organization.....	195
Organizational Unit	195
Originator	165
Originator_System_Title	218, 235
originator-system-title.....	184, 213
OSI model.....	53
OSI-style services	86
Other input	179
Other_Information	235
<i>OtherInfo</i>	183
other-information.....	213
Out of Band.....	191
P/F bit	129
Parameterized access	156
Parameterized_Access	253, 255, 258, 260
<i>PartyUInfo</i>	183
<i>PartyVInfo</i>	183
Passive opening	101
Password	164
P-Data services	364
PH Data transfer services.....	67
PH Layer management services	67
PH-ABORT	67
PH-ABORT service.....	70
PH-ABORT.confirm	70, 84
PH-ABORT.indication	70, 80, 84
PH-ABORT.request	70, 84
PH-ABORT.request / .confirm	80
PH-CONNECT	67
PH-CONNECT service	68
PH-CONNECT.confirm	69, 77, 84
PH-CONNECT.indication	68, 78, 84
PH-CONNECT.request	68, 76, 84
PH-DATA	67
PH-DATA service	69
PH-DATA.indication	70
PH-DATA.request	69
PH-DATA.request / .indication	79
PhL connection establishment/release services	67
PH-Layer service primitives	76, 84
Physical layer, data communications	76
Physical address	62
Physical connection manager	352
Physical connection, setting up	72
Physical device	55, 61
Physical layer	62, 83, 347, 364
Physical layer operation, procedures	71
Physical layer Protocol Data Unit	71
Physical layer services	66, 67, 76
Physical layer, disconnection	76
Physical layer, protocol specification	71
Physical layer, service definitions	68
Physical layer, service specification	67
Physical link, disconnecting	124
Physical link, setting up	123
Ping Service	368
PING service	367
Ping-no-response	369
Ping-system-title-nok	369
PKI architecture	192
Plaintext	169, 172, 185, 209
Point-to-multipoint	66, 110
Point-to-point	66, 110
Poll/final bit	130
Port number	63
Positive Acknowledgement with Retransmission	92, 103
Pre-established application association	227
Presentation layer	153
Priority 157, 236, 239, 242, 247, 282, 286, 287	
Private key	175, 178

Processing_Option	247, 251
Profile specific service parameters	349
Programming mode	82
Proposed_xDLMS_Context.....	226
proposed-conformance.....	273
proposed-dlms-version-number	273
Protection_Element.....	205, 232, 235, 299
Protocol connection parameters	225
Protocol identification service.....	65
Protocol mode E.....	81
Protocol mode E, flow diagram	82
Protocol specification, Phy layer.....	111
Protocol_Connection_Parameters	349, 359
Protocol_Parameters.....	253
protocol-version	267
P-Sync	365
Public attribute	156
Public Client	56, 62, 65, 91, 128, 380, 401, 413
Public key	175, 178
Public key algorithm	32, 168, 175, 189
Public key certificate	190
Public key infrastructure	190
Pull operation	59, 416
Push operation	59, 155, 168, 417
Push setup object.....	270
Random number generation	184
Raw_Data	237, 241, 244, 256, 285
Read service	155, 254, 290
Read.confirm.....	257
Read.indication	257
Read.request	257
Read.response.....	257
Read_Data_Block_Access	253, 255
Readout mode	82
readRequest	257
ReadRequest	291
readResponse	257
ReadResponse.....	256, 291, 292
Real-world IP networks	361
reason.....	268
Receive not ready	131
Receive ready	131
Reception_Threshold	370
Recipient.....	165
Recipient_System_Title	218, 235
recipient-system-title	184, 213
Reference model, S-FSK profile	364
Register service	367
REGISTERED and LOCKED state	377
REGISTERED and UNLOCKED state	377
Registered COSEM names	268
REGISTERED state	378, 379
Registered system	362
Registration.....	55
Remote_IP_Address	88
Remote_TCP_Port	93
Remote_UDP_Port	88
Remote_wPort	88
Repeater status.....	369, 371, 375
RepeaterCall service	367, 369
reply_to_HLS_authentication.....	164
Reporting system	362
Request_Type	236, 239, 242
Reserved wrapper port numbers.....	91, 101
reset_new_not_synchronized	378
Responder.....	101
responder-acse-requirements	273
Responding_AE_Qualifier	225
Responding-AE-Qualifier	273
Responding-AP-Title	273
responding-authentication-value	267, 274
Response time-out (TO_WAIT_RESP).....	144
Response_Type	236, 239
response-allowed	271, 273, 275, 382
Result.....	225, 267, 282
Result (-)	256, 259
Result (+)	256, 259
Result Source-Diagnostic	268
Revision History	26
RLRE APDU	228
RLRE APDU, BER encoding	439
RLRQ APDU	228
RLRQ APDU, BER encoding	438
RNR frame	131
Root Certification Authority.....	192
Root-CA	192
Root-CA certificate	191
RR frame.....	131, 142, 144
SAP Assignment.....	56
Search Initiator Phase	373, 374, 378
search_initiator_threshold	373
search_initiator_time_out	373
Secret key algorithm.....	168
Secret keying material	179, 180, 181
Secure Hash Algorithm	185
Security algorithm ID	184
Security concept	161
Security context	58, 59, 162, 165
Security control byte	186, 208, 209, 213
Security header	208
Security mechanism name	226
Security personalisation	201
Security policy	165, 205
Security setup	53, 191, 202
Security setup\ interface class	179
Security suite	178, 184
Security_Options	205, 232, 299
Security_Status	205, 232, 299
Security_Suite_Id	208
Segmentation	366
Segmentation bit	126, 138
Segmentation, HDLC	138
Segmentation, M-Bus	399
Selective access	53, 155, 245
Self_Descriptive	247, 251
Self-descriptive response	246
Semantic interoperability	65
SEND() function	88
SEND/RECEIVE state	104
Sender ACSE requirements	273

Sequence numbers	104, 130
Server	54, 55, 56, 57, 59, 60, 217
Server system title	273
Server_LLC_Address	349
Server_Max_Receive_PDU_Size.....	227
server_system_title	57
Service Access Point.....	55, 64, 162
Service invocation	279, 302
Service mapping	346, 348
Service primitives.....	221
Service specification	112
Service specification, Phy layer.....	111
Service_Class236, 239, 242, 247, 251, 360, 382	
Service_Class == Unconfirmed	227
Service_Class parameter	227
Service-specific ciphering.....	209
Service-specific dedicated ciphering	210
Service-specific global ciphering	210
Set normal response mode.....	131
SET service.....	155, 238, 283
SET.confirm	241
SET.indication	241
SET.request	241
SET.response	241
SetMapperTable.....	160
SetMapperTable.request	261
Set-Request.....	241
SET-REQUEST-FIRST-BLOCK ..	240, 284, 294
SET-REQUEST-FIRST-BLOCK-WITH-LIST	240, 284, 294
SET-REQUEST-LAST-BLOCK....	240, 284, 294
Set-Request-Normal.....	284
SET-REQUEST-NORMAL ..	240, 284, 294, 298
SET-REQUEST-ONE-BLOCK....	240, 284, 294
Set-Request-With-Datablock	284
SET-REQUEST-WITH-FIRST-BLOCK	240
Set-Request-With-First-Datablock	284
Set-Request-With-List	284
SET-REQUEST-WITH-LIST	240, 284, 294, 298
Set-Request-With-List-And-With-First- Datablock.....	284
Set-Response	241
SET-RESPONSE-ACK-BLOCK...240, 284, 295	
Set-Response-Datablock.....	284
SET-RESPONSE-LAST-BLOCK .240, 284, 295	
SET-RESPONSE-LAST-BLOCK-WITH-LIST	240, 284, 295
Set-Response-Last-Datablock	284
Set-Response-Normal	284
SET-RESPONSE-NORMAL.....	240, 284, 295
Set-Response-With-List.....	284
SET-RESPONSE-WITH-LIST	240, 284, 295
Setting up the data link.....	134
S-FSK PLC environment	56
Shared secret.....	179, 180, 181
Short Name referencing	58, 154, 156
Short_M-Bus_Data_Header.....	394
signatureAlgorithm	194
signatureValue	194
SMS short wrapper.....	413
SNRM command	134, 144
SNRM frame.....	114, 131, 143
Source address	126, 129
Source UDP	91
Source wPort.....	90
Special addresses	129
Start/stop transmission	134
Static key agreement key pair	190
Static key pair	180
Static private key.....	181
Static public key	181
Static Unified Model	179, 189, 482
Streaming.....	159, 300
Sub_Tslot position.....	371
Sub-CA	193
Sub-CA certificate	191
Subject, public key certificate	191
Subordinate Authority.....	192
Sub-slot.....	362
Supporting layer	63
Supporting layer services	346, 348
Symmetric key	186
Symmetric key algorithm	32, 168, 169
Symmetric key block cipher	171
Synchronization.....	365
synchronization_locked	377
Syntactic interoperability	65
System integration.....	65
System title	55, 56
System_Title_Server	368
system-title.....	213
tbsCertificate.....	194
TCP Connected	103
TCP connection	101
TCP connection abort.....	102, 106
TCP connection closing	92, 106
TCP connection establishment	92, 104
TCP connection establishment by the server	361
TCP connection manager process	86, 92
TCP data communication.....	92
TCP disconnection	102
TCP packets.....	100
TCP-ABORT service.....	97
TCP-ABORT.indication	97
TCP-CONNECT	93
TCP-CONNECT.confirm	94
TCP-CONNECT.indication	93
TCP-CONNECT.request	93
TCP-CONNECT.response	94
TCP-DATA service	98, 107
TCP-DATA services.....	103
TCP-DATA.confirm	99
TCP-DATA.indication	98
TCP-DATA.request.....	98
TCP-DISCONNECT services	95
TCP-DISCONNECT.confirm	96
TCP-DISCONNECT.indication	95
TCP-DISCONNECT.request	95

DLMS User Association	2015-12-14	DLMS UA 1000-2 Ed. 8.1	499/500
-----------------------	------------	------------------------	---------

TCP-DISCONNECT.response.....	96
TCP-UDP/IP based communication profile .	28,
61, 355	
Third party	53, 60, 167, 217
Three-letter manufacturer ID	56
Three-way handshake	101, 104
time_out_not_addressed	368
Timeslot	362
TLS-Certificate.....	193
Transaction_Id	235
transaction-id	213
Transfer syntax	58, 379
Transmission Control Protocol.....	85, 92
Transmission credits, M-Bus	403
Transmission order and characteristics	71
Transparency	133
Transport layer address, M-Bus	396, 398
Transporting.....	31
Transporting long messages, 3-layer, CO, HDLC based profile.....	352
Transporting long messages, TCP-UDP/IP based profile	361
TriggerEventNotificationSending	352, 361
TriggerEventNotificationSending service ..	252,
361, 384	
Trust anchor.....	191, 201
Trust model	191
Two-way alternate data transfer	110, 137
UA frame.....	115, 132
UA response	131, 132, 134, 136, 144
UDP Datagram	89
UDP-DATA service	87
UDP-DATA.confirm	89
UDP-DATA.indication	88
UDP-DATA.request	88
UI frame	129, 137, 138, 139, 141, 142
UI frame, sending from server to client.....	141
Unbalanced connection-mode	110
UNC basic repertoire.....	130
Uncompressed form, ECC public key	197
Unconfirmed service.....	154
Unconfirmed service invocations	279
UnconfirmedWrite service	155, 259, 298
UnconfirmedWrite.indication.....	261
UnconfirmedWrite.request	260
unconfirmedWriteRequest	261
Unified service	245
Unilateral authentication.....	162
Unnumbered information (UI) command and response.....	132
Unsolicited service	155
Upper HDLC address	127
User Datagram Protocol	85, 87
User_Information.....	227
user-information	267, 268
V(R)	132
V(S)	132
Valid wPort numbers	89, 99
Validity period	191
Variable Access Specification.....	253
Variable_Access_Specification	155, 255
Variable_Name.....	253, 255, 258, 260
Variable-Access-Specification	254, 258, 260
Virtual circuit	92
Wide Area Network.....	53, 355
Window size.....	145
WINDOW_SIZE	135
wPort.....	357
wPort number	360
Wrapped_Key_Options	235
wrapped-key.....	188
Wrapper	86
Wrapper header	90
Wrapper port	62, 91
Wrapper protocol data unit	90, 100
Wrapper sublayer	86, 89, 100, 357
Wrapper sublayer, state transition diagram	103
Write service	155, 257, 294
Write.confirm	259
Write.indication	259
Write.request	259
Write.response	259
Write_Data_Block_Access	253, 258
writeRequest	259
WriteRequest	294
writeResponse	259
WriteResponse	259, 295
X.509 v3 Certificate	193
xDLMS ASE	60, 152, 154
xDLMS conformance block	159
xDLMS context	58, 59, 153
xDLMS initiate service	154
xDLMS InitiateRequest	186, 226
xDLMS InitiateResponse	186
xDLMS procedures	161
xDLMS services, LN referencing	262
xDLMS version	159
XML document	153
XML schema	270, 324
XML Schema Definitions Language	324
XX-DISCONNECT.request	229