

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**Electricity metering data exchange – The DLMS/COSEM suite –
Part 5-3: DLMS/COSEM application layer**

**Échange des données de comptage de l'électricité – La suite DLMS/COSEM –
Partie 5-3: Couche application DLMS/COSEM**



THIS PUBLICATION IS COPYRIGHT PROTECTED

Copyright © 2013 IEC, Geneva, Switzerland

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

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

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

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

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

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

About the IEC

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

About IEC publications

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

Useful links:

IEC publications search - www.iec.ch/searchpub

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

It also gives information on projects, replaced and withdrawn publications.

IEC Just Published - webstore.iec.ch/justpublished

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

Electropedia - www.electropedia.org

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

Customer Service Centre - webstore.iec.ch/csc

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

A propos de la CEI

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

A propos des publications CEI

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

Liens utiles:

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

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

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

Just Published CEI - webstore.iec.ch/justpublished

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

Electropedia - www.electropedia.org

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

Service Clients - webstore.iec.ch/csc

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

INTERNATIONAL STANDARD

NORME INTERNATIONALE



**Electricity metering data exchange – The DLMS/COSEM suite –
Part 5-3: DLMS/COSEM application layer**

**Échange des données de comptage de l'électricité – La suite DLMS/COSEM –
Partie 5-3: Couche application DLMS/COSEM**

INTERNATIONAL
ELECTROTECHNICAL
COMMISSION

COMMISSION
ELECTROTECHNIQUE
INTERNATIONALE

PRICE CODE
CODE PRIX

XH

ICS 17.220; 35.110; 91.140.50

ISBN 978-2-83220-812-0

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

CONTENTS

FOREWORD.....	7
1 Scope.....	9
2 Normative references	9
3 Terms, definitions and abbreviations	11
3.1 Terms and definitions	11
3.2 Abbreviations	11
4 Overview	13
4.1 COSEM application layer structure	13
4.2 COSEM application layer services	14
4.2.1 ASO services.....	14
4.2.2 Services provided for application association establishment and release	14
4.2.3 Services provided for data transfer	15
4.2.4 Layer management services	19
4.2.5 Summary of COSEM application layer services	19
4.3 COSEM application layer protocols	20
5 Information security in DLMS/COSEM	20
5.1 Definitions	20
5.2 General	20
5.3 Data access security	21
5.3.1 Overview	21
5.3.2 Lowest level security (no security)	21
5.3.3 Low Level Security (LLS)	21
5.3.4 High Level Security (HLS)	22
5.4 Data transport security	23
5.4.1 Applying, removing or checking the protection: ciphering and deciphering	23
5.4.2 Security context	25
5.4.3 Security policy	25
5.4.4 Security suite	25
5.4.5 Security material	26
5.4.6 Ciphered xDLMS APDUs	26
5.4.7 Cryptographic keys	27
5.4.8 The Galois/Counter Mode of Operation (GCM)	30
6 COSEM application layer service specification	40
6.1 Service primitives and parameters.....	40
6.2 The COSEM-OPEN service	42
6.3 The COSEM-RELEASE service	47
6.4 COSEM-ABORT service	49
6.5 Security parameters	50
6.6 The GET service	51
6.7 The SET service.....	53
6.8 The ACTION service	56
6.9 The EventNotification service	60
6.10 The TriggerEventNotificationSending service	61
6.11 Variable access specification	62
6.12 The Read service	63

6.13	The Write service	67
6.14	The UnconfirmedWrite service.....	70
6.15	The InformationReport service.....	71
6.16	Client side layer management services: the SetMapperTable.request.....	72
6.17	Summary of services and LN/SN data transfer service mapping	72
7	COSEM application layer protocol specification	73
7.1	The control function.....	73
7.1.1	State definitions of the client side control function	73
7.1.2	State definitions of the server side control function	75
7.2	The ACSE services and APDUs	76
7.2.1	ACSE functional units, services and service parameters	76
7.2.2	Registered COSEM names	79
7.2.3	APDU encoding rules	81
7.2.4	Protocol for application association establishment.....	81
7.2.5	Protocol for application association release	85
7.3	Protocol for the data transfer services	89
7.3.1	Negotiation of services and options – the conformance block	89
7.3.2	Confirmed and unconfirmed service invocations	90
7.3.3	Protocol for the GET service.....	91
7.3.4	Protocol for the SET service.....	94
7.3.5	Protocol for the ACTION service.....	97
7.3.6	Protocol for the EventNotification service	99
7.3.7	Protocol for the Read service	100
7.3.8	Protocol for the Write service.....	104
7.3.9	Protocol for the UnconfirmedWrite service	108
7.3.10	Protocol for the InformationReport service	109
8	Abstract syntax of ACSE and COSEM APDUs	110
Annex A (normative)	Using the COSEM application layer in various communications profiles	124
Annex B (informative)	AARQ and AARE encoding examples.....	126
Annex C (informative)	Encoding examples: AARQ and AARE APDUs using a ciphered application context.....	140
Annex D (informative)	Data transfer service examples	148
Annex E (informative)	Overview of cryptography	163
Annex F (informative)	Significant technical changes with respect to IEC 62056-53	169
Bibliography		172
Index		174
Figure 1	– Structure of the COSEM Application layers	13
Figure 2	– Summary of COSEM AL services	19
Figure 3	– LLS and HLS authentication	23
Figure 4	– Data transport security in DLMS/COSEM	24
Figure 5	– Ciphered xDLMS APDUs.....	26
Figure 6	– Cryptographic protection of xDLMS APDUs using GCM.....	33
Figure 7	– Service primitives.....	40
Figure 8	– Time sequence diagrams	41
Figure 9	– Partial state machine for the client side control function	74

Figure 10 – Partial state machine for the server side control function.....	75
Figure 11 – MSC for successful AA establishment preceded by a successful lower layer connection establishment.....	82
Figure 12 – Graceful AA release using the A-RELEASE service.....	87
Figure 13 – Graceful AA release by disconnecting the supporting layer	88
Figure 14 – Aborting an AA following a PH-ABORT indication	89
Figure 15 – MSC of the GET service.....	92
Figure 16 – MSC of the GET service with block transfer	92
Figure 17 – MSC of the GET service with block transfer, long GET aborted	94
Figure 18 – MSC of the SET service	95
Figure 19 – MSC of the SET service with block transfer.....	96
Figure 20 – MSC of the ACTION service.....	98
Figure 21 – MSC of the ACTION service with block transfer.....	99
Figure 22 – MSC of the Read service used for reading an attribute.....	102
Figure 23 – MSC of the Read service used for invoking a method.....	103
Figure 24 – MSC of the Read Service used for reading an attribute, with block transfer	104
Figure 25 – MSC of the Write service used for writing an attribute.....	107
Figure 26 – MSC of the Write service used for invoking a method.....	107
Figure 27 – MSC of the Write Service used for writing an attribute, with block transfer	108
Figure 28 – MSC of the Unconfirmed Write service used for writing an attribute.....	109
Figure E.1 – Hash function.....	164
Figure E.2 – Encryption and decryption.....	165
Figure E.3 – Message Authentication Codes (MACs)	166
Table 1 – Clarification of the meaning of PDU Size for DLMS/COSEM	16
Table 2 – Security suites	26
Table 3 – Security control field.....	27
Table 4 – Cryptographic keys and their management.....	30
Table 5 – Plaintext and additional authenticated data	34
Table 6 – Example for ciphered APDUs	37
Table 7 – HLS example with GMAC	39
Table 8 – Codes for AL service parameters	42
Table 9 – Service parameters of the COSEM-OPEN service primitives	43
Table 10 – Service parameters of the COSEM-RELEASE service primitives	47
Table 11 – Service parameters of the COSEM-ABORT service primitives	50
Table 12 – Security parameters	50
Table 13 – Service parameters of the GET service	51
Table 14 – GET service request and response types	52
Table 15 – Service parameters of the SET service.....	54
Table 16 – SET service request and response types.....	55
Table 17 – Service parameters of the ACTION service	57
Table 18 – ACTION service request and response types	58
Table 19 – Service parameters of the EventNotification service primitives	60

Table 20 – Service parameters of the TriggerEventNotificationSending.request service primitive	61
Table 21 – Variable Access Specification	63
Table 22 – Service parameters of the Read service	64
Table 23 – Use of the Variable_Access_Specification variants and the Read.response choices	65
Table 24 – Service parameters of the Write service	68
Table 25 – Use of the Variable_Access_Specification variants and the Write.response choices	68
Table 26 – Service parameters of the UnconfirmedWrite service	70
Table 27 – Use of the Variable_Access_Specification variants	71
Table 28 – Service parameters of the InformationReport service	72
Table 29 – Service parameters of the SetMapperTable.request service primitives	72
Table 30 – Summary of ACSE services	73
Table 31 – Summary of xDLMS services for LN referencing	73
Table 32 – Summary of xDLMS services for SN referencing	73
Table 33 – ACSE functional units, services and service parameters	77
Table 34 – Use of ciphered / unciphered APDUs	80
Table 35 – xDLMS Conformance block	90
Table 36 – GET service types and APDUs	91
Table 37 – SET service types and APDUs	95
Table 38 – ACTION service types and APDUs	98
Table 39 – Mapping between the GET and the Read services	100
Table 40 – Mapping between the ACTION and the Read services	101
Table 41 – Mapping between the SET and the Write services	105
Table 42 – Mapping between the ACTION and the Write service	106
Table 43 – Mapping between the SET and the UnconfirmedWrite services	108
Table 44 – Mapping between the ACTION and the UnconfirmedWrite services	109
Table 45 – Mapping between the EventNotification and InformationReport services	110
Table B.1 – Conformance block	127
Table B.2 – A-XDR encoding of the xDLMS InitiateRequest APDU	128
Table B.3 – A-XDR encoding of the xDLMS InitiateResponse APDU	129
Table B.4 – BER encoding of the AARQ APDU	132
Table B.5 – Complete AARQ APDU	134
Table B.6 – BER encoding of the AARE APDU	135
Table B.7 – The complete AARE APDU	139
Table C.1 – A-XDR encoding of the xDLMS InitiateRequest APDU	140
Table C.2 – Authenticated encryption of the xDLMS InitiateRequest APDU	141
Table C.3 – BER encoding of the AARQ APDU	142
Table C.4 – A-XDR encoding of the xDLMS InitiateResponse APDU	143
Table C.5 – Authenticated encryption of the xDLMS InitiateResponse APDU	144
Table C.6 – BER encoding of the AARE APDU	145
Table C.7 – BER encoding of the RLRQ APDU	146
Table C.8 – BER encoding of the RLRE APDU	147

Table D.1 – Objects used in the examples	148
Table D.2 – Example: Reading the value of a single attribute without block transfer	149
Table D.3 – Example: Reading the value of a list of attributes without block transfer	150
Table D.4 – Example: Reading the value of a single attribute with block transfer	151
Table D.5 – Example: Reading the value of a list of attributes with block transfer	153
Table D.6 – Example: Writing the value of a single attribute without block transfer	155
Table D.7 – Example: Writing the value of a list of attributes without block transfer	156
Table D.8 – Example: Writing the value of a single attribute with block transfer	158
Table D.9 – Example: Writing the value of a list of attributes with block transfer	160

Withdrawing
IECNORM.COM: Click to view the full PDF of IEC 62056-5-3:2013

INTERNATIONAL ELECTROTECHNICAL COMMISSION

**ELECTRICITY METERING DATA EXCHANGE –
THE DLMS/COSEM SUITE –****Part 5-3: DLMS/COSEM application layer****FOREWORD**

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

The International Electrotechnical Commission (IEC) draws attention to the fact that it is claimed that compliance with this International Standard may involve the use of a maintenance service concerning the stack of protocols on which the present standard IEC 62056-5-3 is based.

The IEC takes no position concerning the evidence, validity and scope of this maintenance service.

The provider of the maintenance service has assured the IEC that he is willing to provide services under reasonable and non-discriminatory terms and conditions for applicants throughout the world. In this respect, the statement of the provider of the maintenance service is registered with the IEC. Information may be obtained from:

DLMS¹ User Association
Zug/Switzerland
www.dlms.ch

¹ Device Language Message Specification.

International Standard IEC 62056-5-3 has been prepared by IEC technical committee 13: Electrical energy measurement, tariff- and load control.

This edition cancels and replaces IEC 62056-53 published in 2006. It constitutes a technical revision.

The significant technical changes with respect to IEC 62056-53 are listed in Annex F.

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

FDIS	Report on voting
13/1523/FDIS	13/1541/RVD

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

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

A list of all the parts in the IEC 62056 series, published under the general title *Electricity metering data exchange– The DLMS/COSEM suite*, can be found on the IEC website.

Future standards in this series will carry the new general title as cited above. Titles of existing standards in this series will be updated at the time of the next edition.

The numbering scheme has changed from IEC 62056-XY to IEC 62056-X-Y. For example IEC 62056-53 becomes IEC 62056-5-3.

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

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

IMPORTANT – The 'colour inside' logo on the cover page of this publication indicates that it contains colours which are considered to be useful for the correct understanding of its contents. Users should therefore print this document using a colour printer.

ELECTRICITY METERING DATA EXCHANGE – THE DLMS/COSEM SUITE –

Part 5-3: DLMS/COSEM application layer

1 Scope

This part of IEC 62056 specifies the DLMS/COSEM application layer in terms of structure, services and protocols for COSEM clients and servers, and defines how to use the DLMS/COSEM application layer in various communication profiles.

It defines services for establishing and releasing application associations, and data communication services for accessing the methods and attributes of COSEM interface objects, defined in IEC 62056-6-2², using either logical name (LN) or short name (SN) referencing.

Annex A (normative) defines how to use the COSEM application layer in various communication profiles. It specifies how various communication profiles can be constructed for exchanging data with metering equipment using the COSEM interface model, and what are the necessary elements to specify in each communication profile. The actual, media-specific communication profiles are specified in separate parts of the IEC 62056 series.

Annex B, Annex C and Annex D (informative) include encoding examples for APDUs.

Annex E (informative) provides an overview of cryptography.

2 Normative references

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

IEC 61334-4-41:1996, *Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 41: Application protocols – Distribution line message specification*

IEC 61334-6:2000, *Distribution automation using distribution line carrier systems – Part 6: A-XDR encoding rule*

IEC/TR 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*

² To be published simultaneously with this part of IEC 62056.

IEC 62056-6-1:—, *Electricity metering data exchange – The DLMS/COSEM suite – Part 6-1: Object Identification System (OBIS)*³

IEC 62056-6-2:—, *Electricity metering data exchange – The DLMS/COSEM suite – Part 6-2: COSEM interface classes*⁴

IEC 62056-8-3:—, *Electricity metering data exchange – The DLMS/COSEM suite – Part 8-3: Communication profile for PLC S-FSK neighbourhood networks*⁵

ISO/IEC 15953:1999, *Information technology – Open Systems Interconnection – Service definition for the Application Service Object Association Control Service Element*

ISO/IEC 15954:1999, *Information technology – Open Systems Interconnection – Connection-mode protocol for the Application Service Object Association Control Service Element*

ISO/IEC 8824-1:2008, *Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation*

ISO/IEC 8825-1:2008, *Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)*

FIPS PUB 180-1:2002, *Secure hash standard*

FIPS PUB 197:2001, *Advanced Encryption Standard (AES)*

NIST SP 800-38D:2007, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC*

NIST SP 800-57:2006, *Recommendation for Key Management – Part 1: General (Revised)*

RFC 1321:1992, Internet Engineering Task Force (IETF). *The MD5 Message-Digest Algorithm*. Edited by R. Rivest (MIT Laboratory for Computer Science and RSA Data Security, Inc.) April 1992. Available from: <http://www.rfc-editor.org/rfc/rfc1321.txt>

RFC 3394:2002, Internet Engineering Task Force (IETF). *Advanced Encryption Standard (AES) Key Wrap Algorithm*. Edited by J. Schaad (Soaring Hawk Consulting) and R. Housley (RSA Laboratories) September 2002. Available from: <http://www.rfc-editor.org/rfc/rfc3394.txt>

RFC 4106:2005, *The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)*

NOTE See also the Bibliography.

³ To be published simultaneously with this part of IEC 62056.

⁴ To be published simultaneously with this part of IEC 62056.

⁵ To be published simultaneously with this part of IEC 62056.

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in RFC 4106, IEC/TR 62051 and IEC/TR 62051-1 and the following apply.

3.2 Abbreviations

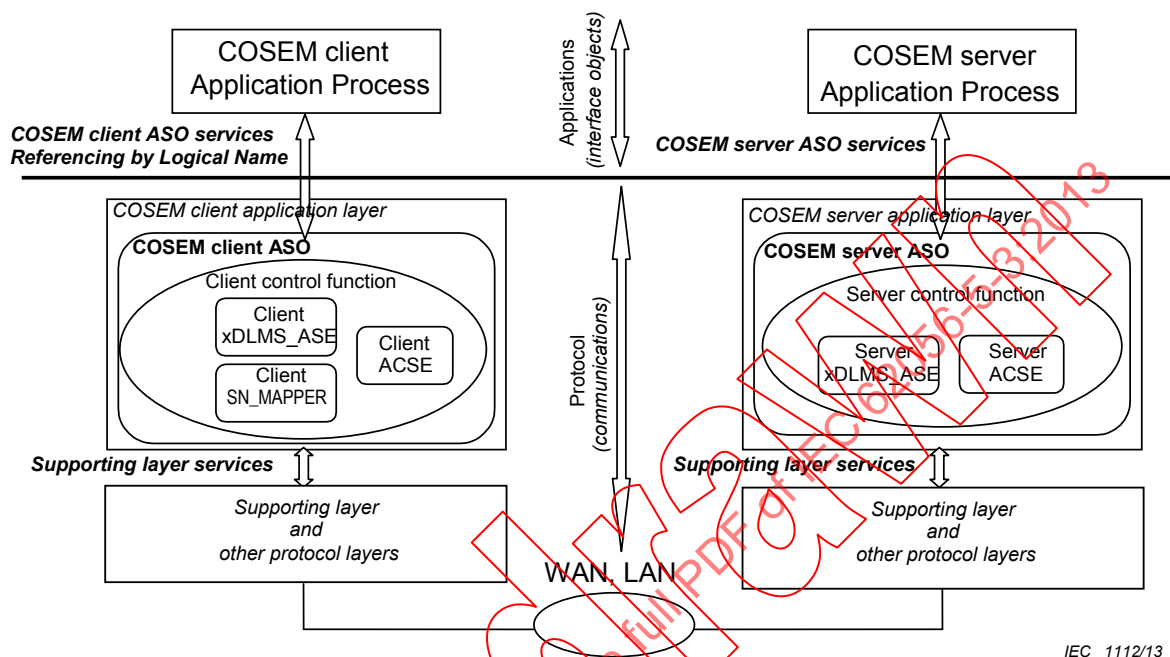
AA	Application Association
AAD	Additional Authenticated Data (used with ciphering of APDUs)
AARE	A-Associate Response – an APDU of the ACSE
AARQ	A-Associate Request – an APDU of the ACSE
ACPM	Association Control Protocol Machine
ACSE	Association Control Service Element
AE	Application Entity
AES	Advanced 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
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
CF	Control Function
CL	Connectionless
Client	A station, asking for services. In the case of the 3-layer, CO HDLC based profile it is the master station
.cnf	.confirm service primitive
CO	Connection-oriented
COSEM	Companion Specification for Energy Metering
COSEM_class_id	COSEM interface class identifier
COSEM object	An instance of a COSEM interface class
DCS	Data Collection System
DLMS	Device Language Message Specification
DLMS UA	DLMS User Association
FCS	Frame Check Sequence
FIPS	Federal Information Processing Standard
GCM	Galois/Counter Mode, 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
HDLC	High-level Data Link Control

HLS	High Level Security
HMAC	Keyed-Hash Message Authentication Code
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
.ind	.indication service primitive
IP	Internet Protocol
ISO	International Organization for Standardization
IV	Initialization Vector
LDN	Logical Device Name
LLC	Logical Link Control (Sublayer)
LLS	Low Level Security
LSB	Least Significant Bit
MAC	Medium Access Control (sublayer)
MAC	Message Authentication Code (cryptography)
master	Central station – station which takes the initiative and controls the data flow
MSB	Most Significant Bit
MSC	Message Sequence Chart
NIST	National Institute of Standards and Technology
OBIS	OBject Identification System
OSI	Open System Interconnection
PDU	Protocol Data Unit
PHSDU	PH SDU
PLC	Power line carrier
PPP	Point-to-Point Protocol
.req	.request service primitive
.res	.response service primitive
RLRE	A-Release Response – an APDU of the ACSE
RLRQ	A-Release Request – an APDU of the ACSE
SAP	Service Access Point
Server	A station, delivering services. The tariff device (meter) is normally the server, delivering the requested values or executing the requested tasks.
TCP	Transmission Control Protocol
TDEA	Triple Data Encryption Algorithm
UDP	User Datagram Protocol
VAA	Virtual Application Association
xDLMS_ASE	Extended DLMS Application Service Element

4 Overview

4.1 COSEM application layer structure

The structure of the client and server COSEM application layers is shown in Figure 1.



IEC 1112/13

Figure 1 – Structure of the COSEM Application layers

The main component of the COSEM AL is the COSEM Application Service Object. It provides services to its service user, the COSEM Application Process, and uses services provided by the supporting lower 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 SN_MAPPER ASE.

The task of the ACSE is to establish, maintain, and release application associations. For the purposes of DLMS/COSEM connection oriented (CO) communication profiles, the CO ACSE, specified in ISO/IEC 15953 and ISO/IEC 15954 is used.

The task of the xDLMS_ASE is to provide data transfer services between COSEM APs. It is based on the DLMS standard, IEC 61334-4-41. It has been extended for DLMS/COSEM; see 4.2.3.1. The main objective of DLMS/COSEM is to provide a business domain oriented interface object model for metering devices and systems while keeping backward compatibility with the DLMS standard. To meet these objectives, DLMS/COSEM includes an evolution of DLMS. Remaining fully compliant to the DLMS standard, DLMS/COSEM provides a more metering specific view of the meter through the COSEM interface objects.

IEC 62056-6-2:—, 4.2 specifies two referencing methods to attributes and methods of COSEM interface objects for use in COSEM servers: LN and SN referencing. When LN referencing is used, the Logical Name of the COSEM objects shall be as specified in IEC 62056-6-1. Therefore, on the server side, two distinct xDLMS service sets are specified: one exclusively

for LN referencing and the other exclusively for SN referencing. It can be considered that there are two different xDLMS_ASEs: one providing services for LN referencing and the other for SN referencing. The server AL may include one, the other or both xDLMS_ASEs.

The 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.

NOTE Both the client and the server COSEM ASO can contain other, optional application protocol components.

The optional 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 also 6.17.

The COSEM AL also performs some functions of the OSI presentation layer:

- encoding the ACSE APDUs in BER (ISO/IEC 8825-1). See also 7.2.3;
- encoding the xDLMS APDUs carrying the data transfer services in A-XDR (IEC 61334-6).

4.2 COSEM application layer services

4.2.1 ASO services

The services required by, or provided for the COSEM client and server APs at the logical interfaces with the respective COSEM AL, using CO procedures, fall into three categories:

- application association establishment and release;
- data transfer;
- layer management.

4.2.2 Services provided for application association establishment and release

The services provided for the establishment and release of AAs 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 used 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. 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. Therefore,

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 6.2, the COSEM-RELEASE service in 6.3 and the COSEM-ABORT service in 6.4.

4.2.3 Services provided for data transfer

4.2.3.1 The xDLMS application service element

As mentioned in 4.1, xDLMS includes some extensions to the DLMS standard, IEC 61334-4-41. These extensions define added functionality; existing functionality is not modified. They are made in such a way, that there is no conflict with the existing DLMS standard and comprise the following:

- additional services;
- additional data types;
- new DLMS version number;
- new conformance block;
- clarification of the meaning of the PDU Size.

The additional services are GET, SET, ACTION and EventNotification. They allow to reference attributes and methods of COSEM interface objects using LN referencing. See 4.2.3.5 and 4.2.3.6.

The additional data types are specified in Clause 8

The new DLMS version number, corresponding to the first version of the xDLMS ASE is 6.

The new 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 7.3.1 and Clause 8.

For services using SN referencing, new variants of the Variable_Access_Specification service parameter, the Read.response and the Write.response services have been added to support selective access and block transfer.

To clarify the meaning of the maximum PDU size usable by the client and the server, the modifications shown in Table 1 have been made. The xDLMS-Initiate service uses these names for PDU Sizes.

Table 1 – Clarification of the meaning of PDU Size for DLMS/COSEM

was:	new:
Page 61, Table 3, 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, 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 PDUs. The value proposed in an Initiate request shall be large enough to always permit the Initiate Error PDU transmission.	The Client Max Receive PDU Size parameter, of type Unsigned16, contains the maximum length expressed in bytes for a DLMS PDU that the server may send. The client will discard any received PDUs that are longer than this maximum length. The value shall be large enough to always permit the AARE APDU transmission. Values below 12 are reserved. The value 0 indicates that there is no limit on the PDU size.
Page 63, last paragraph, 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 PDUs. A PDU that is longer than this maximum length will be discarded. This maximum length is computed as the minimum of the Proposed Max PDU Size and the maximum PDU size than the VDE-handler may support.	The Server Max Receive PDU Size parameter, of type Unsigned16, contains the maximum length expressed in bytes for a DLMS PDU that the client may send. The server will discard any received PDUs that are longer than this maximum length. Values below 12 are reserved. The value 0 indicates that there is no limit on the PDU size.

4.2.3.2 Client/server and non-client/server type services

xDLMS data transfer services are related to attributes and methods of COSEM interface objects, specified in IEC 62056-6-2 and provide the interface between the COSEM application model and the communication protocols.

When the server uses SN referencing, attributes and methods of COSEM interface objects are mapped to DLMS named variables. The mapping is held by the Association SN interface object(s).

For accessing attributes and methods, client/server type services are used: the client requests services and the server provides them.

There is also an unsolicited, non-client/server type service. This service is requested by the server, upon occurrence of an event, to inform the client of the value of one or more attributes, as though they had been requested by the client. It is an unconfirmed service.

4.2.3.3 Referencing methods and service mapping

As specified in 4.1, there are two distinct service sets available, one for LN and one for SN referencing.

On the client side, in order to handle the different referencing methods transparently for the AP, the AL provides only one service set, using LN referencing. Using a unique, standardized service set between COSEM client APs and the communication protocol – hiding the particularities of COSEM servers using different referencing methods – allows to specify 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, services using LN referencing and/or SN referencing can be provided.

In the case of confirmed AAs, the referencing method and the service set to be used are negotiated during the AA establishment phase via the COSEM application context; see 7.2.2.2, and the conformance block; see 7.3.1. 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 and the service set supported by the server.

When the server uses LN referencing, the services are the same on both side. When the server does not use LN referencing, a mapping between client side and server side services takes place. As explained in 4.1, if the server uses SN referencing this mapping is performed by the Client SN_MAPPER ASE. See also 6.17.

4.2.3.4 Confirmed and unconfirmed services

Within confirmed AAs, client/server type data transfer services can be invoked in a confirmed or unconfirmed manner. Within unconfirmed AA-s, client/server type data transfer services may be invoked in an unconfirmed manner only. See also 7.3.2.

4.2.3.5 COSEM client/server type services

COSEM client/server type data transfer services for LN referencing are the following:

- the GET service, used to read the value of one or more attributes of COSEM interface objects. See 6.6;
- the SET service, used to write the value of one or more attributes of COSEM interface objects. See 6.7;
- the ACTION service, used to invoke one or more methods of COSEM interface objects. Invoking methods may imply sending service parameters and returning data. See 6.8.

COSEM client/server type data transfer services for SN referencing are the following:

- the Read service, used to read the value of one or more attributes or to invoke one or more methods of COSEM interface objects when return parameters are expected. It is a confirmed service. See 6.10;
- the Write service, used to write the value of one or more attributes or to invoke one or more methods of COSEM interface objects when no return parameters are expected. It is a confirmed service. See 6.13;
- the UnconfirmedWrite service, used to write the value of one or more attributes or to invoke one or more methods of COSEM interface objects. It is an unconfirmed service. See 6.14.

4.2.3.6 Services for event notification

To support event notification, COSEM also provides non client/server type, unsolicited services:

- with LN referencing the EventNotification service. See 6.9;
- with SN referencing, the InformationReport service. See 6.15.

4.2.3.7 Identifying 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. 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.

The EventNotification service – being a non-client/server type service – does not contain the `Invoke_Id` parameter.

This feature is available only with LN referencing.

4.2.3.8 Priority of service invocations: the Priority parameter

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 shall be served before the previous requests with priority NORMAL. The response shall carry the same priority flag as that of the corresponding request. Managing priority is a negotiable feature; see 7.3.1.

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.

4.2.3.9 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. 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 these access selection parameters. In the case of LN referencing, this feature is called Selective access; see 6.6 and 6.7. It is a negotiable feature; see 7.3.1. In the case of SN referencing, this feature is called Parameterized access; see 0, 6.13 and 6.14. It is a negotiable feature; see 7.3.1.

4.2.3.10 Multiple references

In a data transfer service invocation, it is possible to reference one or several attributes respectively methods. Using multiple references is a negotiable feature. See 7.3.1.

4.2.3.11 Attribute_0 referencing

With the GET and SET services, a special feature, `Attribute_0` referencing, is available. By convention, attributes of COSEM interface objects are numbered from 1 to n, where `Attribute_1` is the logical name of the COSEM interface 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 6.6 and with the SET service in 6.7. `Attribute_0` referencing is a negotiable feature; see 7.3.1.

NOTE As specified in IEC 62056-6-2:—, 4.1 and 4.2, manufacturers can add proprietary methods and/or attributes to any object, using negative numbers.

4.2.3.12 Transferring long service parameters

The data transfer services are carried by the APDUs, exchanged between the peer ALs, in an encoded form. In some cases, the data may be longer than the Client / Server Max Receive PDU Size negotiated. To transfer such 'long' data, two transporting mechanisms are available:

- long data transfer using the AL block transfer protocol. This mechanism can be used with any confirmed client/server type services. It is a negotiable feature. See 7.3.1;
- long data transfer in a transparent manner to the AL. This feature can be used when the supporting layer(s) provide(s) segmentation; see Annex A.

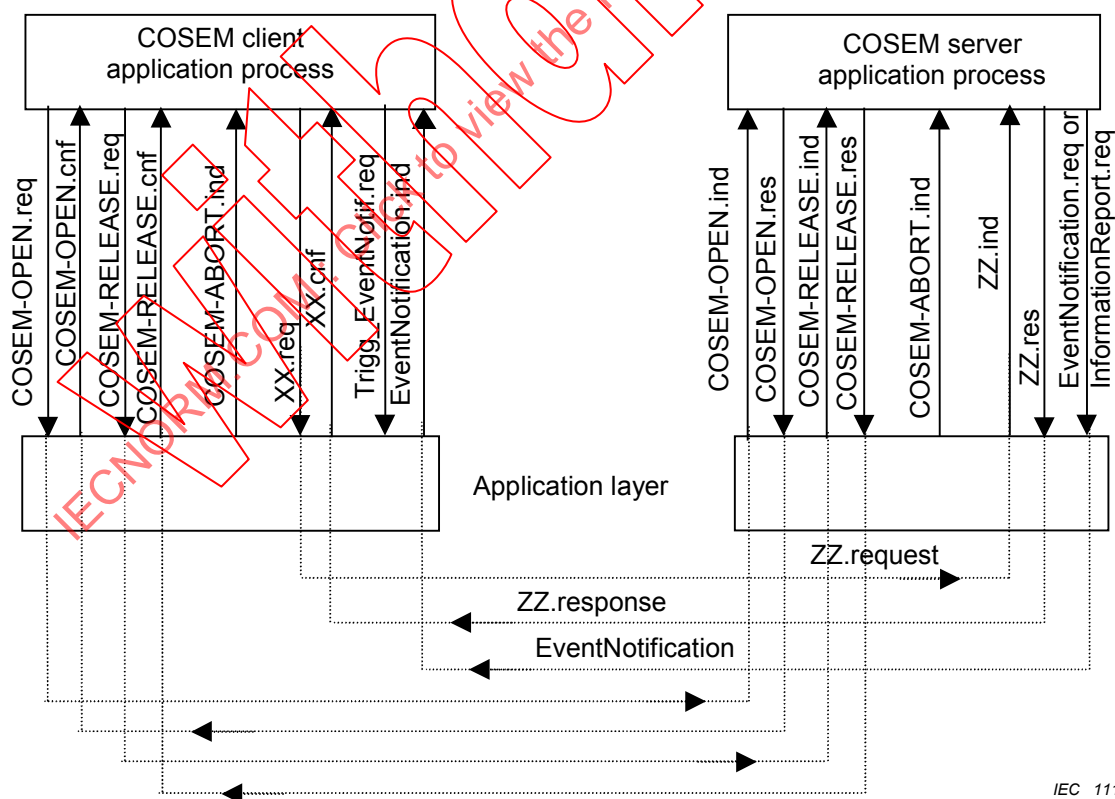
NOTE There is no mechanism within AL block transfer for re-sending or requesting a missing or corrupt block. Such a mechanism can be a feature of long data transfer managed transparently by the supporting layers.

4.2.4 Layer management services

Layer management services have local importance only. Therefore, specification of these services is not within the scope of this companion specification. The specific SetMapperTable service is defined in 6.16.

4.2.5 Summary of COSEM application layer services

A summary of the services available at the top of the COSEM AL is shown in Figure 2. Layer management services are not shown. Although the service primitives are different on the client and server side, the APDUs are the same. The abstract syntax of the ACSE and COSEM APDUs is specified in Clause 8.



IEC 1113/13

NOTE XX and ZZ refer to client/server type data transfer services. These services are different on the client side and the server side, if the server does not use LN referencing. See 6.17.

Figure 2 – Summary of COSEM AL services

4.3 COSEM application layer protocols

The 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 AL protocol is based on the ACSE and the DLMS protocols, as specified in ISO/IEC 15954 and in IEC 61334-4-41 – with the extensions for DLMS/COSEM – respectively. 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 abstract syntax of the APDUs using ASN.1 as specified in ISO/IEC 8824-1.

The COSEM AL protocols are specified in Clause 7.

5 Information security in DLMS/COSEM

NOTE Clause 5 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.

5.1 Definitions

5.1.1

confidentiality

preserving authorized restrictions on information access and disclosure, including means for protecting personal privacy and proprietary information. A loss of confidentiality is the unauthorized disclosure of information

[SOURCE: FIPS PUB 199]

5.1.2

integrity

guarding against improper information modification or destruction, and includes ensuring information non-repudiation and authenticity. A loss of integrity is the unauthorized modification or destruction of information

[SOURCE: FIPS PUB 199]

5.2 General

Confidentiality and integrity are among the key requirements when open systems connect to public media. With the increase of computational power, the requirements for strong cryptographic methods also increase. DLMS/COSEM provides two main information security features for accessing and transporting data:

- *data access security* controls access to the data held by a DLMS/COSEM server. See 5.3;
- *data transport security* allows the sending party to apply cryptographic protection to the xDLMS APDUs sent to ensure confidentiality and integrity. This requires ciphered ADPUs. The receiving party can remove or check this protection. See 5.4.

These information security features are provided partly by the COSEM AL, partly by COSEM objects:

- upon AA establishment, two contexts are negotiated using the ACSE services: the *application context* and the *authentication context*. The application context determines whether ciphered APDUs can be used or not. The authentication context determines the level of data access security. The ACSE APDUs are not cryptographically protected;

The user-information field of the ACSE APDUs shall be cryptographically protected as determined by the application-context and the security policy.

- once an AA is successfully established, COSEM services can be used to access attributes and methods of COSEM objects, depending on the access rights valid in the given association. These services are carried by xDLMS APDUs, which may be cryptographically protected: authenticated, encrypted, or both, depending on the security policy in force. The cryptographic protection is applied, removed or checked by the COSEM AL. The COSEM AP of the sending party informs, via service parameters, the COSEM AL about the protection to be applied to the APDU to be sent. The COSEM AL of the receiving party informs the COSEM AP about the protection that has been applied to the APDU received.

The message security methods described in this part of IEC 62056 have been selected from the standards specified by NIST and the IETF.

For an overview of cryptography; see Annex E.

5.3 Data access security

5.3.1 Overview

Data access security concerns role based access to data in a DLMS/COSEM device.

It is managed by the Association LN / Association SN objects. Each COSEM server i.e. a logical device may support AAs with various clients, each having a different role, and with this, different access rights. Each AA is identified with a pair of lower layer addresses. Each Association object provides a list of objects visible in that particular AA and the access rights to their attributes and methods.

To be able to access data, the client shall be properly authenticated. Upon AA establishment, an authentication context is negotiated between the client and the server. This specifies the required authentication of the peers, and, where needed, the security algorithm to verify the authentication. Three data access security levels are provided:

- Lowest level security (no security);
- Low Level Security (LLS);
- High Level Security (HLS).

5.3.2 Lowest level security (no security)

The purpose of lowest level security is to allow the client to retrieve some basic information. This authentication context does not require any peer authentication; it allows direct access to the data in the server, within the access rights available in the given AA.

5.3.3 Low Level Security (LLS)

The purpose of Low Level Security is to allow the authentication of clients by verifying the password supplied. The server is not authenticated. This authentication context is typically used when the communication channel offers adequate security to avoid eavesdropping and message (password) replay.

The client has to supply the correct password during the process of AA establishment. If the password is OK, the AA is established and the client can access data within the access rights available in the given AA. Otherwise, the AA is not established.

The LLS authentication process is supported by the COSEM-OPEN service of the COSEM AL – see 6.2 – and it is as follows:

- the client transmits a “secret” (for example a password) to the server, by using the “Calling_Authentication_Value” parameter of the COSEM-OPEN.request service primitive;
- the server checks if the “secret” received corresponds to the client identification;
- if yes, the client is authenticated and the AA can be established. If no, the AA is rejected;
- the result with diagnostic information is sent back by the server using the COSEM-OPEN.response service primitive.

The association objects provide a method/attribute to change the “secret”; see IEC 62056-6-2:—, 5.3.1, 5.3.2.

The authentication process with LLS is shown on the left side of Figure 3.

5.3.4 High Level Security (HLS)

The purpose of High Level Security is to allow mutual authentication of the client and the server participating in an association. This authentication context is typically used when the communication channel offers no intrinsic security and precautions have to be taken against eavesdroppers and against message (password) replay.

HLS requires that both the client and the server mutually authenticate each other. This is a 4-pass process, involving the exchange of challenges during AA establishment, which is followed by exchanging the results of processing these challenges, using cryptographic methods. If the authentication takes place, the client can proceed to access data within the access rights available in the given AA, and it accepts data coming from the server. Otherwise, the AA is not established.

The HLS authentication process is supported by the COSEM-OPEN service – see 6.2 – and by the Association LN / SN objects as follows:

- Pass 1: The client transmits a “challenge” CtoS – for example, a random string – to the server;
- Pass 2: The server transmits a “challenge” StoC – for example, a random string – to the client;

The length of the challenges shall be 8 octets to 64 octets.

Pass 3: The client processes a “challenge” StoC according to the rules of the HLS authentication mechanism negotiated:

- in the case of HLS authentication_mechanism_id(2), the method of processing the challenge is secret – for example encrypting with a secret key – which is the HLS secret known by both the client and the server;
- in the case of HLS authentication_mechanism_id(3), the client right concatenates StoC received during Pass2 with the HLS secret, known by both the client and the server, and generates a digest using the MD5 algorithm as specified in RFC 1321;
- in the case of HLS authentication_mechanism_id(4), the process is the same, but the client generates a digest using the SHA-1 algorithm. FIPS PUB 180-1 applies in this respect;
- in the case of HLS authentication_mechanism_id(5), an authentication tag is calculated using GMAC. See 5.4.8.4.

The result – f(StoC) – is sent back 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: If the client is authenticated, the server processes the “challenge” CtoS in the same way as described in Pass3. The result – f(CtoS) – is sent back 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.

The authentication process with HLS is shown on the right side of Figure 3.

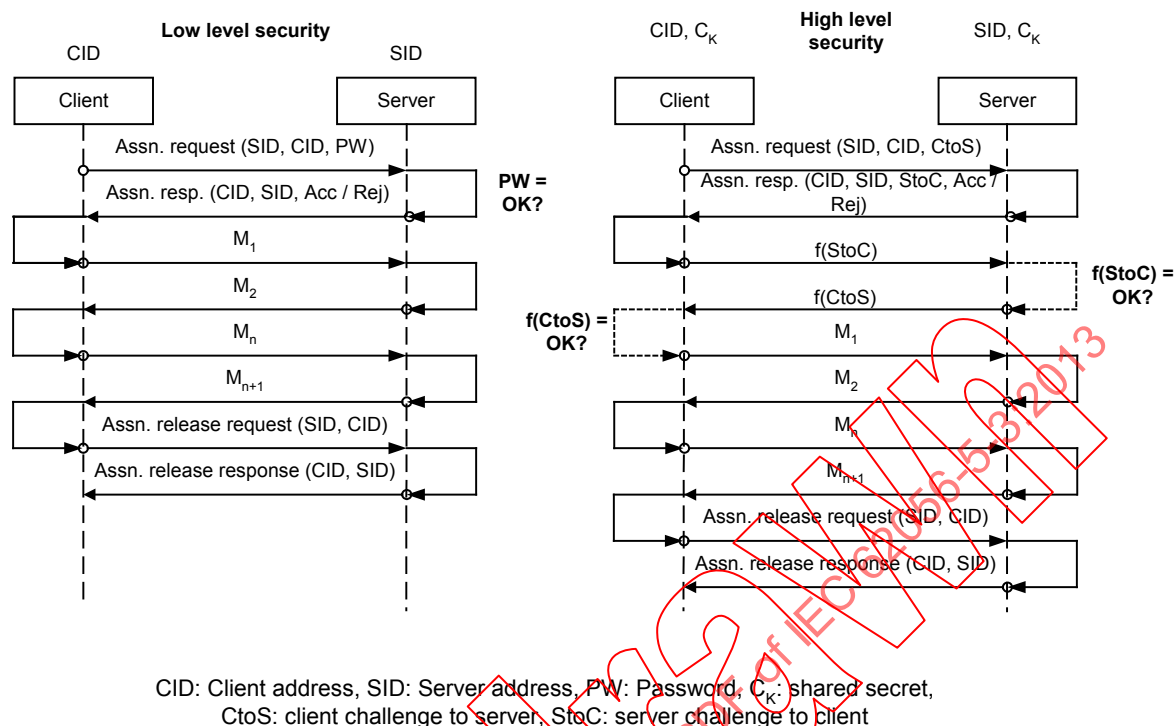


Figure 3 – LLS and HLS authentication

Pass 1 is supported by the COSEM-OPEN.request service primitive of the client AL. The parameter "Security_Mechanism_Name" carries the identifier of the HLS mechanism, and the parameter "Calling_Authentication_Value" carries the challenge CtoS.

Pass 2 is supported by the COSEM-OPEN.response service primitive of the server AL. The parameter "Security_Mechanism_Name" carries the identifier of the HLS mechanism, and the parameter "Responding_Authentication_Value" carries the challenge StoC.

After Pass 2, the AA is formally established, but the access of the client is restricted to the method "reply_to_HLS_authentication" of the current "Association" object.

Pass 3 and Pass 4 are supported by the method reply_to_HLS_authentication of the association object(s); see IEC 62056-6-2:—, 5.3.1 and 5.3.2. If both passes are successfully executed, then full access is granted according to the current AA. Otherwise, either the client or the server aborts the AA.

In addition, the Association SN / LN object provides the method to change the HLS "secret": change_HLS_secret.

REMARK After the client has issued the change_HLS_secret () – or change_LLS_secret () – method, it expects a response from the server acknowledging that the secret has been changed. It is possible that the server transmits the acknowledgement, but due to communication problems, the acknowledgement is not received at the client side. Therefore, the client does not know if the secret has been changed or not. For simplicity reasons, the server does not offer any special support for this case; i.e. it is left to the client to cope with this situation.

5.4 Data transport security

5.4.1 Applying, removing or checking the protection: ciphering and deciphering

For data transport security, cryptographic protection can be applied before sending an xDLMS APDU – as determined by the security policy; see 5.4.3 – then removed or checked after the reception of an APDU.

NOTE Cryptographic protection of data in storage is out of the scope of this companion specification.

Applying the protection is achieved by ciphering. Deciphering removes or checks the protection, restoring the original xDLMS APDU. Ciphering and deciphering is performed by the COSEM AL as shown in Figure 4.

When a COSEM service request or response primitive is invoked by the client or the server AP respectively, the service parameters include the Security_Options parameter. This parameter informs the AL on the requested security to be applied on the xDLMS APDU carrying the service primitive, and it may identify elements of the security material to be used.

Similarly, a COSEM indication or confirm service primitive includes a Security_Status parameter. This parameter informs the AP on the security that was applied on the xDLMS APDU carrying the service primitive, and it may include information on the elements of the security materials used. The authentication tag – if any – is also passed.

For COSEM service specifications, see Clause 6.

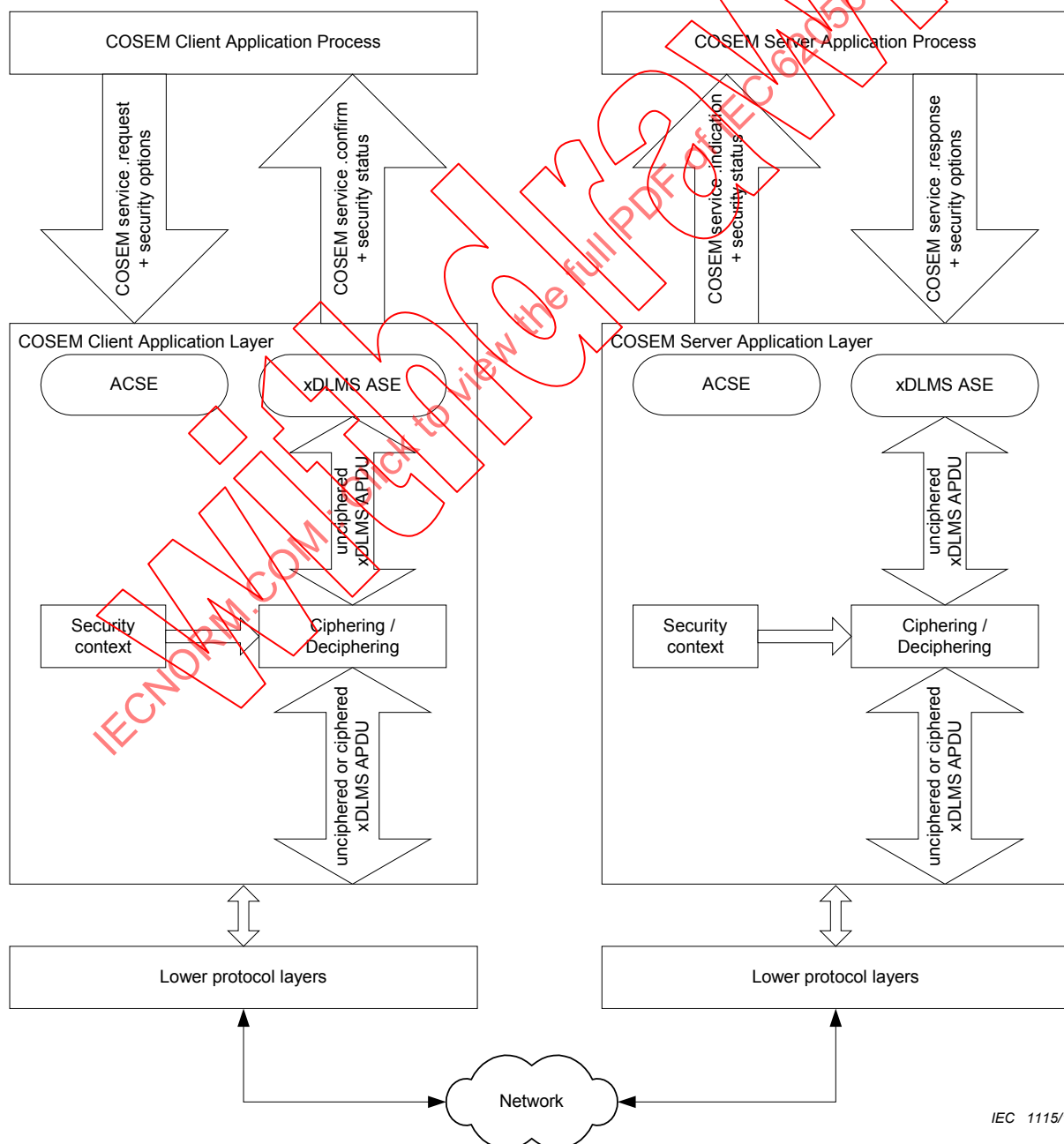


Figure 4 – Data transport security in DLMS/COSEM

5.4.2 Security context

The security context defines security attributes relevant for the ciphering / deciphering process:

- the security policy in force, determining the kind of protection to be applied. See 5.4.3;
- the security suite, specifying the security algorithm(s). See 5.4.4;
- the security material relevant for the given security suite, including elements like block cipher keys, authentication keys, initialization vectors etc. See 0.

5.4.3 Security policy

The following security policies are specified:

- security is not imposed;
- all messages are authenticated;
- all messages are encrypted;
- all messages are authenticated and encrypted.

The security policy is held by the security_policy attribute of the Security setup object; see IEC 62056-6-2:—, 5.3.5.

Authenticated access may be required selectively: certain attributes and methods may be accessible only by using authenticated messages. This is indicated by the access mode to each attribute and method; see IEC 62056-6-2:—, 5.3.1 and 5.3.2 respectively. Therefore, authenticated xDLMS APDUs may be used – within a ciphered application context – even when the security policy in effect does not require that all messages be authenticated.

Note that if authenticated access is required, but the application context is not a ciphered one, the attributes and methods requiring authentication cannot be accessed. If the client attempts to send a ciphered APDU in an unciphered application context, then the server application layer shall reject it. In the case of SN referencing, the response shall be ConfirmedServiceError APDU, carrying the tag of the rejected APDU. In the case of LN referencing, the response shall be an ExceptionResponse APDU.

On the other hand, encryption applies generally for all messages within a given AA: if the security policy requires that all messages shall be encrypted, then all APDUs shall be encrypted – and additionally they may be authenticated.

When the security policy requires that all messages shall be authenticated and encrypted, then only APDUs that are both encrypted and authenticated can be used.

Messages protected by higher security than what the security policy requires are always allowed (provided that the application context negotiated allows them).

5.4.4 Security suite

A security suite determines the cryptographic algorithm(s) used for message security. A security suite is identified with a Security Suite ID; see Table 2. Currently, one security suite is specified, the Galois/Counter Mode (GCM) with AES-128. In this security suite, global keys – see 5.4.7.3.3 – are protected during transportation using the AES-128 key wrap algorithm.

Table 2 – Security suites

Security Suite Id	Authentication algorithm	Encryption algorithm	Key transport method
0	AES-GCM-128	AES-GCM-128	Key wrapping using AES-128 key wrap
All other reserved	–	–	–

5.4.5 Security material

The elements of the security material are the following:

- a block cipher key, denoted EK ;
- an authentication key, denoted AK ;
- an initialization vector, denoted IV .

The elements depend on the security suite. For AES-GCM-128, they are defined in 5.4.8.

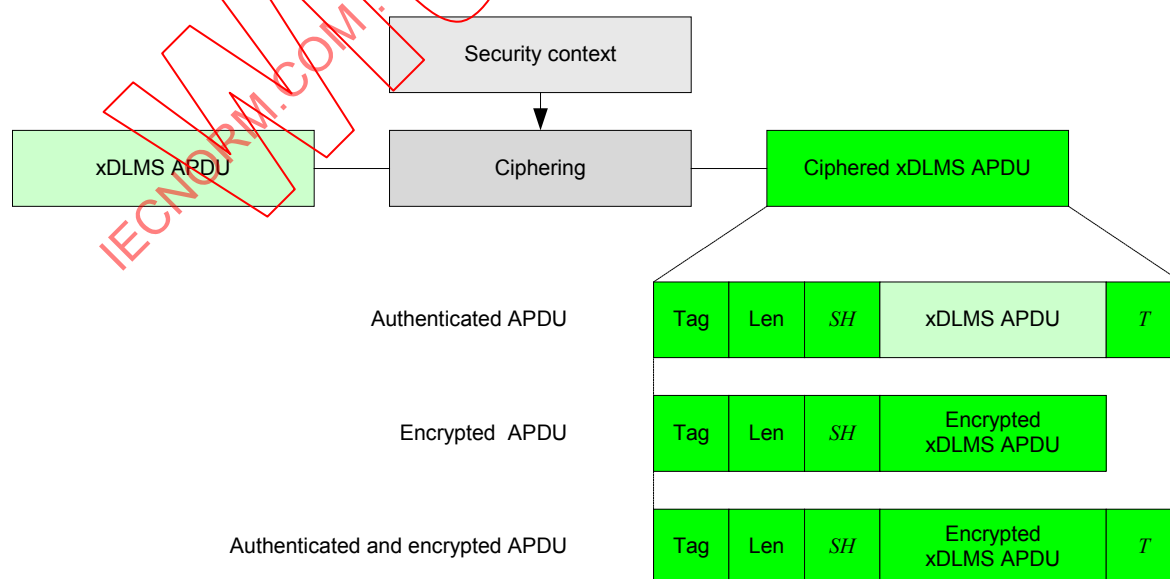
5.4.6 Ciphred xDLMS APDUs

Ciphred xDLMS APDUs can be used in a ciphred COSEM application context only. A ciphred APDU is always an octet string.

In a ciphred COSEM application context, unciphred APDUs may also be used.

The structure of the ciphred APDU depends on the kind of ciphering applied; see Figure 5:

- if the APDU is authenticated only, it contains the APDU tag, the length field, the security header field, the original APDU and the authentication tag;
- if the APDU is encrypted only, it contains the APDU tag, the length field, the security header field, and the encrypted DLMS APDU;
- if the APDU is both authenticated and encrypted, it contains the APDU tag, the length field, the security header field, the encrypted APDU and the authentication tag.



IEC 1116/13

Figure 5 – Ciphred xDLMS APDUs

The APDU tag identifies the type of the APDU; see Clause 8.

The length field specifies the length of the octet-string (including the security header, the unencrypted or encrypted xDLMS APDU and the authentication tag).

The Security Header, *SH*, includes the Security Control field, *SC* right concatenated with the Frame Counter, *FC*.

The Security Control field is shown in Table 3, where:

- Bit 3...0: Security_Suite_Id; see 5.4.4;
- Bit 4: “A” subfield: indicates that the APDU is authenticated;
- Bit 5: “E” subfield: indicates that the APDU is encrypted;
- Bit 6: Key_set subfield 0 = Unicast;
1 = Broadcast; see 5.4.7.2;
- Bit 7: Reserved, shall be set to 0.

Table 3 – Security control field

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3...0
Reserved, set to 0	Key_set	E	A	Security_Suite_Id

The Frame Counter is an internal counter maintained by the sending and receiving parties.

Finally, *T* is the authentication tag, calculated by the authentication algorithm.

5.4.7 Cryptographic keys

5.4.7.1 General

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. For the purposes of DLMS/COSEM, examples of operations include:

- the transformation of plaintext data into ciphertext data;
- the transformation of ciphertext data into plaintext data;
- the computation of an authentication code from data;
- the verification of an authentication code from data and a received authentication code.

5.4.7.2 Key types

Various key types are defined. They are identified according to their classification, their cryptographic function, their use and their scope.

By their classification, a distinction is made between:

- *symmetric* keys. A symmetric key is a single cryptographic key that is used with a secret (symmetric) key algorithm. For an overview of symmetric key algorithms; see E.3;
- *public* and *private* keys, used with public key cryptographic algorithms. For an overview of asymmetric key algorithms; see E.4.

For symmetric keys, a distinction is made between:

- *Symmetric authentication keys*, used with symmetric key algorithms to provide assurance of the integrity and source of messages;
- *Symmetric data encryption keys*, used with symmetric key algorithms to apply confidentiality protection to information; and

- *Symmetric key wrapping keys*, used to encrypt other keys using symmetric key algorithms. Key wrapping keys are also known as key encrypting keys (KEK).

By their use, a distinction is made between:

- *unicast keys*, used to secure unicast communication between two peers;
- *broadcast keys*, used to secure broadcast communication between a DLMS/COSEM client and several DLMS/COSEM servers. In DLMS/COSEM, broadcast can be initiated by the client only.

By their scope, a distinction is made between *dedicated keys* and *global keys*. These terms are DLMS/COSEM specific:

- a *dedicated key* is a ciphering key that may be used to cipher xDLMS APDUs from the establishment of an AA between a client and a server until the AA is released. Therefore, its lifetime is the same as the lifetime of the AA. It is delivered during AA establishment and used in subsequent transmissions within the same AA.

When the AA is released, the dedicated key should be destroyed by the server as well as by the client.

- a *global key* is a ciphering key that may be (re-)used to cipher xDLMS APDUs, exchanged between the same client and server, in more than one instance of the AA.

The following (symmetric) encryption keys may be available: global unicast key, global broadcast key, dedicated unicast key.

A (symmetric) authentication key is always a global key that can be used in unicast and broadcast messages.

In Addition, security suite specific rules may apply. See also Table 4.

5.4.7.3 Key management

5.4.7.3.1 System architectures

NOTE Subclause 5.4.7.3 provides general guidelines for secure implementation. The management of the security system, including the security keys is the responsibility of the system operator.

For the purposes of 5.4.7.3, two general system architectures are considered:

- the COSEM client is the central DCS, exchanging data directly with COSEM servers;
- the COSEM client is a concentrator, exchanging data on the one hand with COSEM servers, and on the other hand with the central DCS. In this case, the concentrator may or may not be a COSEM server at the same time. The central DCS may or may not be a COSEM client.

For generation and distribution of symmetric keys, NIST SP 800-57:2006, 8.1.5.2 applies.

5.4.7.3.2 Master key

A master key shall be present in each COSEM server logical device configured in the system. This key is used for wrapping global keys; see 5.4.7.3.3.

NOTE 1 The way of generating and installing the master key is out of the scope of this companion specification.

A secure mechanism shall be in place to ensure that the central data DCS knows the master key of each COSEM server logical device participating in the system.

NOTE 2 The specification of such a mechanism is out of the scope of this companion specification.

The concentrators, if they are present in the system, shall not know the master keys.

For security reasons, the lifetime of the master key should be limited. The way to replace the master keys is out of the scope of this standard. For guidance, see NIST SP 800-57:2006, 8.2.3, Key change function.

5.4.7.3.3 Global keys

Before ciphering can be used, global keys have to be generated and delivered to the COSEM logical devices participating in the system.

Global keys shall be generated by the central DCS.

The way to generate the global keys is outside the scope of this companion specification. For security reasons, their lifetime should be limited.

For delivery, global keys shall be wrapped using the AES-128 key wrap algorithm as specified in RFC 3394, and using the master key as the Key Encryption Key (KEK).

When the system works without (a) concentrator(s), the global keys shall be delivered by the central DCS invoking the `global_key_transfer` method of the Security setup interface object in the COSEM server with the wrapped global key as a parameter; see IEC 62056-6-2:—, 5.3.5.

When the system works with (a) concentrator(s), the global keys shall be delivered – wrapped by the master key – to the concentrator. The `global_key_transfer` method of the Security setup interface object in the COSEM server shall be invoked then by the concentrator with the wrapped global key as a parameter. For details, see IEC 62056-6-2:—, 5.3.5.

In any case, the global key is wrapped by the master key.

The concentrator should not know the master keys, therefore it should not be able to wrap / unwrap global keys.

The global keys shall also be made available by the central DCS to the concentrator(s), if present, in a secure way.

NOTE The specification of such a process is out of the scope of this part of IEC 62056.

5.4.7.3.4 Dedicated keys

When the system works without (a) concentrator(s), dedicated keys shall be generated by the central DCS. When the system works with (a) concentrator(s), dedicated keys shall be generated by the concentrators.

For delivery, they shall be included in the dedicated-key field of the xDLMS InitiateRequest APDU, carried by the user-information field of the AARQ APDU. The xDLMS InitiateRequest APDU shall be authenticated and encrypted using the AES-GCM-128 algorithm, the global unicast encryption key and – if in use, see 5.4.8.3.4.4, – the authentication key. Likewise, the xDLMS InitiateResponse APDU, carried by the user-information field of the AARE APDU shall be also encrypted and authenticated the same way. See also 5.2.

The various cryptographic key types, together with their use and their management are summarized in Table 4.

Table 4 – Cryptographic keys and their management

Key type	Use	Generation	Delivery	Location
Symmetric keys				
Master key	Key Encryption Key (KEK) for global keys	Not specified	Not specified	Central DCS and COSEM server
Global unicast encryption key	Global ciphering of unicast xDLMS APDUs	Central DCS	Wrapped with master key, invocation of global_key_transfer method by the central DCS or the concentrator	Central DCS (and Concentrator) and COSEM server
Global broadcast encryption key	Global ciphering of broadcast xDLMS APDUs	Central DCS	Wrapped with master key, invocation of global_key_transfer method by the central DCS or the concentrator	Central DCS (and Concentrator) and COSEM server
Authentication key (Global)	Authentication of xDLMS APDUs	Central DCS	Wrapped with master key, invocation of global_key_transfer method by the central DCS or the concentrator	Central DCS (and Concentrator) and COSEM server
Dedicated (unicast) encryption key	Dedicated ciphering of unicast xDLMS APDUs	Central DCS or Concentrator	Transported as part of the xDLMS InitiateRequest APDU, which is encrypted and authenticated using the AES-GCM-128 algorithm, the global unicast encryption key and the authentication key	Central DCS (or Concentrator) and COSEM server (during the lifetime of the AA)
NOTE If alternatives are given, the first alternative is valid for systems not using concentrators and the second alternative for systems using concentrators.				

5.4.8 The Galois/Counter Mode of Operation (GCM)

5.4.8.1 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. 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 accidental modifications of the data and 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.

For more information, see NIST SP 800-38D.

5.4.8.2 Definitions, abbreviations, symbols and notation relevant for the Galois/Counter mode

Term / Abbreviation	Symbol	Meaning
Additional authenticated data (AAD)	A	Input data to the authenticated encryption function that is authenticated but not encrypted. It is also known as Associated Data.
Authenticated decryption		Function of GCM in which the ciphertext is decrypted into the plaintext, and the authenticity of the ciphertext and the AAD are verified.
Authenticated encryption		Function of GCM in which the plaintext is encrypted into the ciphertext, and an authentication tag is generated on the AAD and the ciphertext.
Authentication key	AK	Part of the AAD
Block cipher		Parameterized family of permutations on bit strings of a fixed length; the parameter that determines the permutation is a bit string called the key.
Ciphertext	C	Encrypted form of the plaintext
	EK	Block cipher key
Fixed field		In the deterministic construction of IVs, field that identifies the device or context for the instance of the authenticated encryption function.
	FC	Frame counter.
Fresh		For a newly generated key, property of being unequal to any previously used key.
GCM		Galois/Counter Mode
Initialization vector	IV	Nonce that is associated with an invocation of authenticated encryption on a particular plaintext and AAD.
Invocation field		In the deterministic construction of IVs, field that identifies the sets of inputs to the authenticated encryption function in a particular device or context. For the purposes of this companion specification, the invocation field is the frame counter.
Key		Parameter of the block cipher that determines the selection of the forward cipher function from the family of permutations.
	KEK	Key encryption key
	$len(X)$	Bit length of the bit string X
	$LEN(X)$	Octet length of the octet string X
Nonce		Value that is used only once within a specified context.
Plaintext	P	Input data to the authenticated encryption function that is both authenticated and encrypted.
	SC	Security control byte
	SH	Security header; concatenation of the security control byte SC and the frame counter FC : $SH = SC \parallel FC$.
System title	$Sys-T$	A unique identifier of the system.
Authentication tag	T	Cryptographic checksum on data that is designed to reveal both the accidental errors and the intentional modification of the data.
	t	Bit length of the authentication tag NOTE This is the same as $len(T)$.
	$X \parallel Y$	Concatenation of two strings, X and Y .

5.4.8.3 Elements of GCM

5.4.8.3.1 Block cipher and keys

For the purposes of this companion specification, the AES encryption standard has been selected. GCM uses the forward cipher function; it does not employ the inverse cipher function.

GCM uses a single key, the block cipher key (the key, for short). Its size shall be 128 bits. In DLMS/COSEM, for additional security, an authentication key is also specified. It is part of the additional authenticated data, AAD. The block cipher key is denoted *EK* and the authentication key is denoted *AK*.

5.4.8.3.2 GCM functions

5.4.8.3.2.1 General

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. 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.

5.4.8.3.2.2 Authenticated encryption – input and output data

NOTE The following text is based on NIST SP 800-38D:2007, 5.2.1.1 and RFC 4106:2005, 2.1.

The authenticated encryption operation has four inputs, each of which shall be an octet string:

- a plaintext, denoted *P*;
- additional authenticated data (AAD), denoted *A*. It shall include – together with other elements – the authentication key, denoted *AK*.
- a secret block cipher key denoted *EK*;
- an initialization vector 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 5.4.8.3.4.5.

There are two outputs:

- a ciphertext, denoted *C* whose length is exactly that of the plaintext *P*;
- an authentication tag, denoted *T*.

5.4.8.3.2.3 Authenticated decryption – input and output data

The authenticated decryption operation has five inputs:

- the secret block cipher key, denoted EK ;
- 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 document.

The output P indicates that T is the correct authentication tag for IV , A , and C ; otherwise, the output is *FAIL*.

The authenticated encryption and authenticated decryption functions, their inputs and outputs, as well as the structure of the secured APDUs are shown in Figure 6 below.

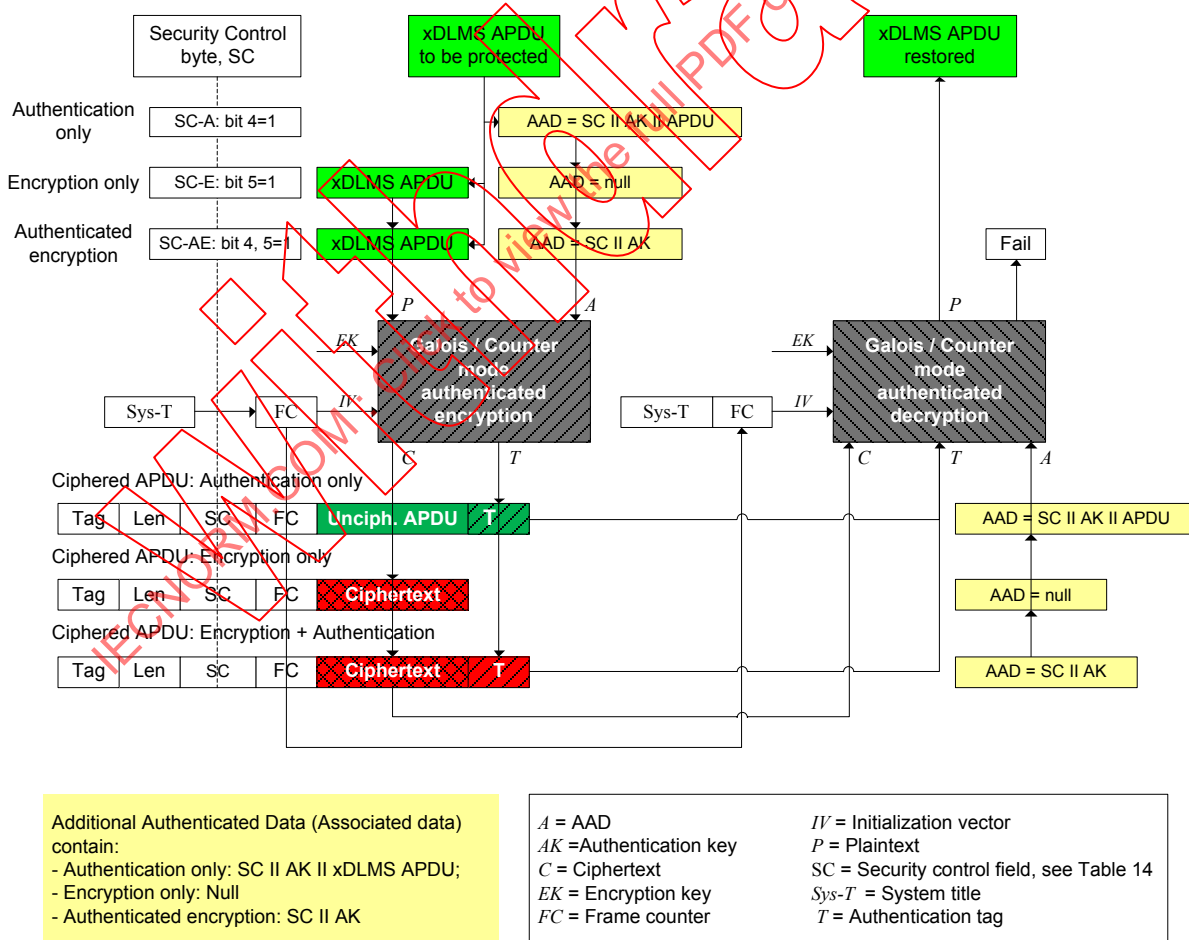


Figure 6 – Cryptographic protection of xDLMS APDUs using GCM

5.4.8.3.3 Security header with GCM

The security header shall be as specified in 5.4.6, with security_suite_id = 0 and with the appropriate values of the Key_set, E and A sub-fields.

5.4.8.3.4 GCM parameters and data elements

5.4.8.3.4.1 Length of the authentication tag

The bit length of the authentication tag, denoted t , is a security parameter. Its value shall be 96 bits.

5.4.8.3.4.2 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 5, where SC is the security control byte, AK is the authentication key and M is the message, i.e. the xDLMS APDU to be protected.

The bit lengths of P and A shall meet the following requirements.

- $\text{len}(P) < 2^{39}-256$;
- $\text{len}(A) < 2^{64}-1$;
- the bit lengths of P and A shall all be multiples of 8, so that these values are octet strings.

Table 5 – Plaintext and additional authenticated data

Security control, SC		Security	P	A
E field	A field			
0	0	None	–	–
0	1	Authenticated only	–	$SC \parallel AK \parallel M$
1	0	Encrypted only	M	–
1	1	Encrypted and authenticated	M	$SC \parallel AK$

5.4.8.3.4.3 Block cipher key, EK

The size of the block cipher key, denoted EK shall be 128 bits (16 octets): $\text{len}(EK) = 128$.

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.

5.4.8.3.4.4 Authentication key, AK

Although not required by GCM, an authentication key denoted AK is specified for additional security in DLMS/COSEM. For its length and its generation, the same rules apply as for the encryption key. The authentication key shall be part of the additional authenticated data AAD. If an authentication key has been transferred, it shall be used, until a null authentication key is transferred.

5.4.8.3.4.5 Initialization vector, IV

For the construction of the initialization vector, IV , deterministic construction as specified in NIST SP 800-38D 8.2.1 shall be used.

In the deterministic construction, 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, two distinct physical devices shall not share the same fixed field, and two distinct sets of inputs to any single device shall not share the same invocation field.

The length of the IV shall be 96 bits (12 octets): $len(IV) = 96$. Within this:

- the leading (i.e. the leftmost) 64 bits (8 octets) shall hold the fixed field. It shall contain the system title; see 5.4.8.3.4.6;
- the trailing (i.e. the rightmost) 32 bits shall hold the invocation field; see 5.4.8.3.4.7.

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.

5.4.8.3.4.6 System title

Each physical device in the system, including the meters, the concentrators when used, and the DCS shall be uniquely identified with its system title. The system title denoted as *Sys-T* shall be constructed as follows:

- its length shall be 8 octets;
- the leading (i.e., the 3 leftmost) octets shall hold the three-letter manufacturer ID⁶. This is the same as used in the leading octets of the Logical Device Name of server logical devices;

The three-letter manufacturer ID shall be used to identify all client and server systems.

NOTE 1 It can be obtained from the DLMS UA.

- the trailing (i.e. the 5 rightmost) octets, together with the manufacturer ID, shall ensure the uniqueness of the system title.

NOTE 2 These 5 octets can be derived for example from the last twelve digits of the manufacturing number, up to 999 999 999 999. This value converts to 0xE8D4A50FFF. Values above this, up to 0xFFFFFFFF (decimal value 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 for the system title, but in any case, the system title shall be 8 octets long – if necessary, right padded to 8 octets – and shall be unique. The details shall be specified by the naming authority designated as such for the project.

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

- when the S-FSK PLC profile is used, system titles are exchanged during the registration process using the CIASE protocol. See IEC 62056-8-3;
- in all communication profiles, system titles are exchanged during AA establishment with a ciphered context using the COSEM-OPEN service – see 6.2 – carried the AARQ / AARE APDUs;

⁶ Administered by the DLMS UA acting as a registration authority; the same as the one used in the LDN.

If the system titles sent / received during AA establishment are not the same as the ones exchanged during the CIASE registration process, the AA shall be rejected.

- by writing the client_system_title attribute and by reading the server_system_title attribute of Security setup objects; see IEC 62056-6-2:—, 5.3.5.

NOTE 3 For broadcast communication, only the COSEM client sends it to the COSEM server.

5.4.8.3.4.7 Invocation field

The invocation field shall be an integer counter (frame counter).

5.4.8.3.5 Example

The following example (see Table 6) illustrates the process of creating ciphered xDLMS APDUs when applying authentication only, encryption only and authenticated encryption.

Withdrawing
IECNORM.COM: Click to view the full PDF of IEC 62056-5-3 © IEC:2013

Table 6 – Example for ciphered APDUs

	<i>X</i>	Contents	<i>LEN</i> (<i>X</i>) bytes	<i>Len</i> (<i>X</i>) 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 hexadecimal form)	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>	000102030405060708090A0B0C0D0E0F	16	128
Authentication Key	<i>AK</i>	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	16	128
Security applied		Authentication	Encryption	Authenticated encryption
Security control byte (with unicast key)	<i>SC</i>	<i>SC-A</i> 10	<i>SC-E</i> 20	<i>SC-AE</i> 30
Security header	<i>SH</i>	<i>SH</i> = <i>SC-A</i> <i>FC</i> 1001234567	<i>SH</i> = <i>SC-E</i> <i>FC</i> 2001234567	<i>SH</i> = <i>SC-AE</i> <i>FC</i> 3001234567
Inputs		Authentication	Encryption	Authenticated encryption
xDLMS APDU to be protected	<i>APDU</i>	C0010000080000010000FF0200 (Get-request, attribute 2 of the Clock object)		
Plaintext	<i>P</i>	Null	C0010000080000010000FF0200	C0010000080000010000FF0200
Associated data	<i>A</i>	<i>SC</i> <i>AK</i> <i>APDU</i>		<i>SC</i> <i>AK</i>
Associated Data – Authentication	<i>A-A</i>	10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDFC0010000080000010000FF0200	–	–
Associated Data – Encryption	<i>A-E</i>	–	–	–
Associated Data – Authenticated encryption	<i>A-AE</i>	–	–	30D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF
Outputs		Authentication	Encryption	Authenticated encryption
Ciphertext	<i>C</i>	NULL	411312FF935A47566827C467BC	411312FF935A47566827C467BC
Authentication tag	<i>T</i>	06725D910F9221D263877516	–	7D825C3BE4A77C3FCC056B6B
The complete Ciphered APDU		<i>TAG</i> <i>LEN</i> <i>SH</i> <i>APDU</i> <i>T</i>	<i>TAG</i> <i>LEN</i> <i>SH</i> <i>C</i>	<i>TAG</i> <i>LEN</i> <i>SH</i> <i>C</i> <i>T</i>
Authenticated APDU		C81E1001234567C0010000080000010000FF020006725D910F9221D263877516	–	–
Encrypted APDU		–	C8122001234567411312FF935A47566827C467BC	–
Authenticated and encrypted APDU		–	–	C81E3001234567411312FF935A47566827C467BC7D825C3BE4A77C3FCC056B6B

5.4.8.4 High Level Security peer authentication with GMAC (authentication_mechanism_id(5))

NOTE See also 5.3.4.

In the case of authentication_mechanism_id(5), the process is the following:

- Pass 1 and Pass 2 shall be the same as described in 5.3.4;
- in Pass 3 and 4, instead of the HLS secret (held by the Association LN / Association SN objects), the global unicast encryption key, and the authentication key (if in use) are used. These are transferred to the server using the global_key_transfer method of the Security setup class. See IEC 62056-6-2:—, 5.3.5;
- in Pass 3, the client processes the challenge StoC. The authentication tag T is calculated using GMAC on $SC \parallel AK \parallel StoC$. The result $f(StoC) = SC \parallel FC \parallel T$ is sent back to the server. The server checks if $f(StoC)$ is the result of correct processing and – if correct – accepts the authentication of the client;
- in Pass 4: if the client is authenticated, the server processes CtoS in the same way as described in Pass 3. The authentication tag T is calculated using GMAC on $SC \parallel AK \parallel CtoS$. The result $f(CtoS) = SC \parallel FC \parallel T$ is sent back to the server. The client checks if $f(CtoS)$ is the result of correct processing and – if correct – accepts the authentication of the server.

An example is shown in Table 7 below.

Table 7 – HLS example with GMAC

Security material	<i>X</i>	Contents		<i>LEN(X)</i> bytes	<i>len(X)</i> bits
Security suite		GCM-AES-128			
System Title	<i>Sys-T</i>	Client	Server		
		4D4D4D0000000001	4D4D4D0000BC614E		
		(here, the five last octets contain the manufacturing number in hexadecimal form)		8	64
Frame Counter	<i>FC</i>	00000001	01234567	4	32
Initialization Vector	<i>IV</i>	Sys-T FC			
		Client	Server	12	96
		4D4D4D0000000001 00000001	4D4D4D0000BC614E 01234567		
Block cipher key (global)	<i>EK</i>	000102030405060708090A0B0C0D0E0F		16	128
Authentication Key	<i>AK</i>	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF		16	128
Security control	<i>SC</i>	10		1	8
		Pass 1: Client sends challenge to server			
CtoS		4B35366956616759 "K56VagY"		8	64
		Pass 2: Server sends challenge to client			
StoC		503677524A323146 "P6wRJ21F"		8	64
		Pass 3: Client processes StoC			
SC AK StoC		10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF503677524A323146			
T (SC AK StoC)		1A52FE7DD3E72748973C1E28		12	96
f(StoC)		10000000011A52FE7DD3E72748973C1E28		17	136
		Pass 4: Server processes CtoS			
SC AK CtoS		10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF4B35366956616759			
T (SC AK CtoS)		FE1466AFB3DBCD4F9389E2B7		12	96
f(CtoS)		1001234567FE1466AFB3DBCD4F9389E2B7		17	136

6 COSEM application layer service specification

6.1 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 7 illustrates this notion of service hierarchy and shows the relationship of the two correspondent N-users and their associated N-layer peer protocol entities.

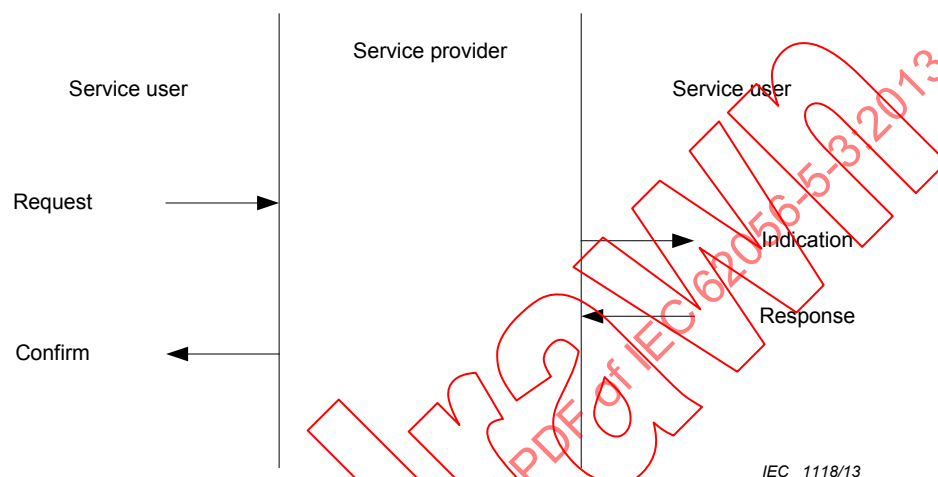


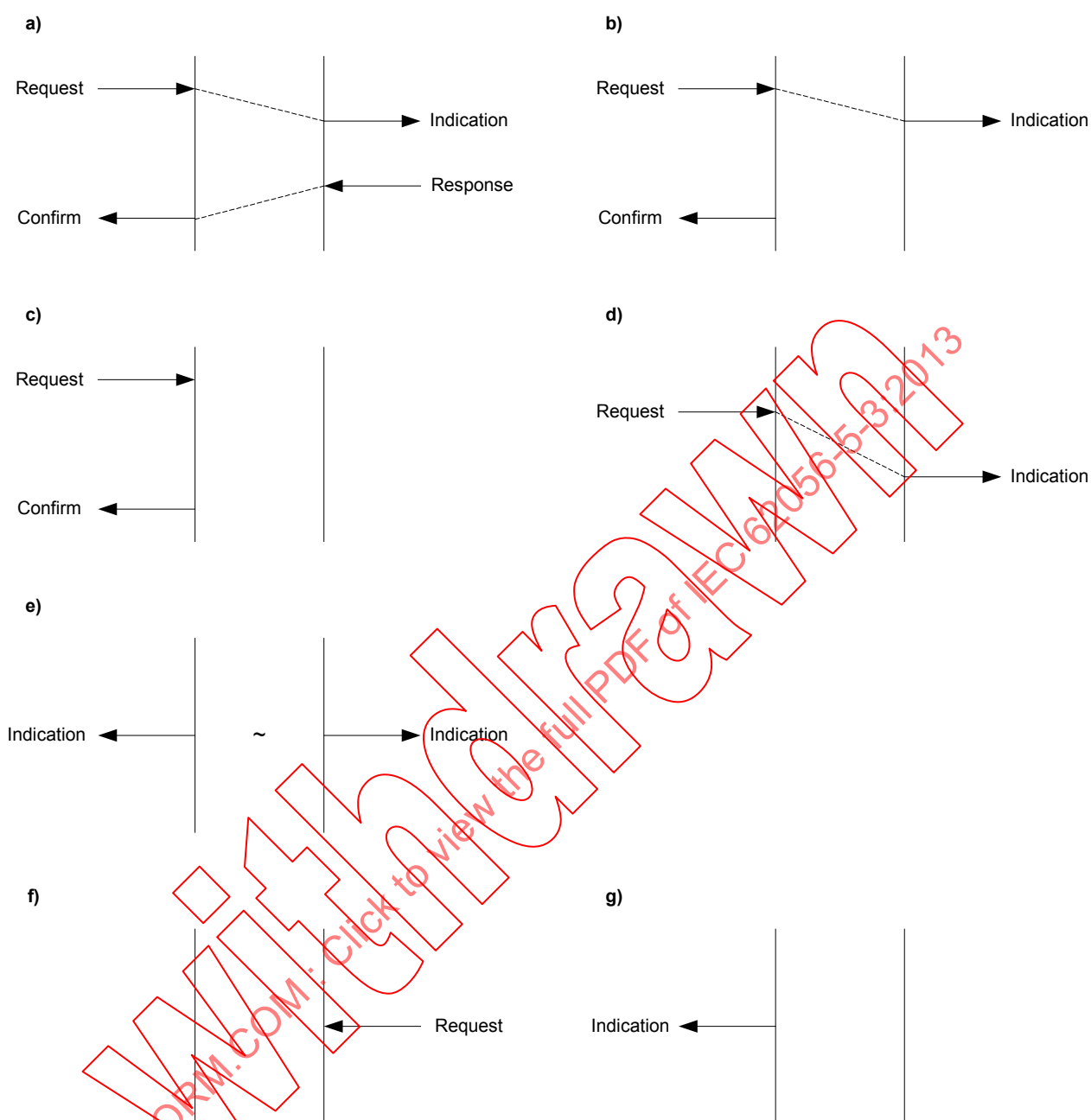
Figure 7 – Service primitives

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 8. 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.



IEC 1119/13

Figure 8 – Time sequence diagrams

The service parameters of the 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 8.

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 8 – 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 standard, 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: 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**;
- direct reference to a service parameter uses the capitalized form, while indirect (non-specific) reference uses the normal text 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.

6.2 The COSEM-OPEN service

NOTE The description has been amended to cover the new data transport security features. The order of the service parameters has been aligned with the order of the fields of the AARQ / AARE APDUs.

Function

The function of the COSEM-OPEN service is to establish an AA between peer COSEM APs. 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 9.

Table 9 – 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 (=)	–	–
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 (=)
XLMS_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 7.2. Encoding examples are given in Annex B.

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 Calling_AP_Title parameter is conditional. When the Application_Context_Name indicates an application context using ciphering, it shall carry the client system title specified in 5.4.8.3.4.6.

The use of the Called_AP_Title, Called_AE_Qualifier, Called_AP_Invocation_Identifier, Called_AE_Invocation_Identifier, Calling_AE_Qualifier, Calling_AP_Invocation_Identifier and Calling_AE_Invocation_Identifier parameters is optional. Their use is not specified in this companion specification.

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 shall carry the server system title specified in 5.4.8.3.4.6.

The use of the Responding_AE_Qualifier, Responding_AP_Invocation_Identifier and Responding_AE_Invocation_Identifier parameters is optional. Their use is not specified in this companion specification.

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 7.2.1.

The presence of the ACSE_Requirements parameter depends on the security mechanism used:

- in the case of Lowest Level security, it shall not be present; only the Kernel functional unit will be used;

- in the case of Low Level security (LLS) it shall be present in the .request primitive and it may be present in the .response service primitive;
- in the case of High Level Security (HLS), it shall be present both in the .request and the .response service primitives.

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 companion specification.

The `Proposed_xDLMS_Context` parameter holds the elements of the proposed xDLMS context 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-128 algorithm, the global unicast encryption key and the authentication key (if in use). The xDLMS InitiateRequest APDU shall be ciphered the same way, when the dedicated key is not present, but it is necessary to protect the RLRQ APDU by including the ciphered xDLMS InitiateRequest in its user-information field; see 6.3.

The `Proposed_DLMS_Version_Number` element holds the proposed DLMS version number; see 4.2.3.1.

The `Proposed_DLMS_Conformance` element holds the proposed conformance block; see 7.3.1.

The `Client_Max_Receive_PDU_Size` element holds the maximum length of the xDLMS APDUs the client can receive; see Table 1.

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 holds the elements of the negotiated xDLMS context, 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 4.2.3.1.

The `Negotiated_DLMS_Conformance` element holds the negotiated conformance block. See 7.3.1.

The `Server_Max_Receive_PDU_Size` element carries the maximum length of the xDLMS APDUs the server can receive; see Table 1.

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 Annex A.

The Service_Class parameter is mandatory. It indicates whether the service shall be invoked in a confirmed or in an unconfirmed manner. The handling of this parameter may depend on the communication profile; see Annex A.

Use

A possible logical sequence of the COSEM-OPEN service primitives is illustrated in Figure 8:

- 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 AP⁷. 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 previously requested 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 7.2.4. Communication profile specific rules are specified in Annex A.

⁷ Before the invocation of the COSEM-OPEN.request primitive, the physical layers shall be connected. Depending on the communication profile, the invocation of this primitive may also imply the connection of other lower layers.

6.3 The COSEM-RELEASE service

NOTE 1 The description has been amended to better align with the A-Release service and to allow the user-information field of the RLRQ / RLRE to carry a ciphered xDLMS InitiateRequest / InitiateResponse APDU, to secure 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 10.

Table 10 – 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 7.2. The Use_RLRQ_RLRE parameter in the .response primitive is conditional. If it was present in the .indication primitive and if 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 2 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, authenticated and encrypted 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, authenticated and encrypted 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 companion specification. See also Annex A.

Use

A possible logical sequence of the COSEM-RELEASE service primitives is illustrated in Figure 8:

- 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.

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 incorrect 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 7.2.5. Communication profile specific rules are specified in Annex A. See also 7.2.

6.4 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 11.

Table 11 – 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 companion specification.

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 an unsolicited 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 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 7.2.5.3.

6.5 Security parameters

When ciphering is used, the COSEM data transfer services carry additional security parameters, as shown in Table 12. These parameters are common for all services.

Table 12 – Security parameters

	.request	.indication	.response	.confirm
Security_Options	C	–	C	–
Security_Status	–	C	–	C
Service parameters depending on the kind of service: – GET, SET, ACTION; – Read, Write, UnconfirmedWrite		...	...	...
Authentication_Tag	–	C	–	C
NOTE The service primitives available depend on the kind of the service.				

The Security_Options parameter is conditional. It shall contain the following information:

- the security control field; see Table 3;
- whether global or dedicated ciphering shall be used (this information determines the tag of the ciphered APDU).

The .request and .response service primitives shall use the same security suite and the same kind of ciphering (global or dedicated).

The Security_Status parameter is conditional. It shall contain the following information:

- the security control field;
- whether global or dedicated ciphering has been used (this information is derived from the tag of the ciphered APDU).

The Authentication_Tag parameter is conditional. It is present only if the APDU has been authenticated or authenticated and encrypted. It carries the authentication tag calculated using the given security suite and keys.

6.6 The GET service

NOTE The following specification of the services using LN referencing has been amended for a coherent presentation with that of the services using SN referencing. No technical changes have been introduced.

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 interface 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 13.

Table 13 – 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 (=)	–	–
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 12.				

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 Annex A.

The use of the Request_Type and Response_Type parameters is shown in Table 14.

Table 14 – 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 can 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 4.2.3.10);
- the Access_Selection_Parameters is present only when COSEM_Object_Attribute_Id != 0 and if selective access to the given attribute is available; see 4.2.3.9. The Access_Selector and Access_Parameters sub-parameters are defined in the COSEM interface object definitions; see IEC 62056-6-2.

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. The Result is of type DataBlock_G. It carries block transfer control information and raw-data:

- the Last_Block element indicates whether the current block is the last one (TRUE) or not (FALSE);
- the Block_Number element carries the number of the actual block sent;
- the (inner) Result element carries either Raw_Data or Data_Access_Result. Within this:
 - if the value of a single attribute was requested, Raw_Data carries a part of the value of the attribute (Data). If the Data cannot be delivered, the response shall 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_Results, 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 7.3.3.

Use

A possible logical sequence of the GET service primitives is illustrated in Figure 8:

- 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 always be 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 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 7.3.3.

6.7 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 interface 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 15.

Table 15 – 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 (=)	–	–
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 12.				

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 Annex A.

The use of the Request_Type and Response_Type parameters is shown in Table 16.

Table 16 – 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 can 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 4.2.3.10);
- the Access_Selection_Parameters is present only when COSEM_Object_Attribute_Id != 0 and if selective access to the given attribute is available; see 4.2.3.9. The Access_Selector and the Access_Parameters sub-parameters are defined in the COSEM interface object definitions; IEC 62056-6-2.

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 referenced attributes. 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. 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 information “success” 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 information “success” 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

A possible logical sequence of the SET service primitives is illustrated in Figure 8:

- 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 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 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 7.3.4.

6.8 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 interface 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.)

Semantics

The ACTION service primitives shall provide parameters as shown in Table 17.

Table 17 – 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 (=)	–	–
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 12.				

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 Annex A.

The use of the Request_Type and Response_Type parameters is shown in Table 18.

Table 18 – 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	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 can 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 the invocation of any of the methods does not require additional parameters, it shall be nevertheless present, but it shall be null data.

- 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 (or null data). The number and the order of the method invocation parameters shall be the same as that of the COSEM attribute descriptors.

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. 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 of COSEM method descriptors. It consists of two elements:

- the Result parameter. It contains either the information “success” 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. 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. If no Response_Parameters are returned, the response shall be of type ACTION-RESPONSE-NORMAL;
 - If a list of methods was invoked, Raw_Data carries a part of the list of Action_Responses 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

A possible logical sequence of the ACTION service primitives is illustrated in Figure 8:

- 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 interface objects of the server AP. If the complete list of COSEM method descriptors and method invocation parameters fits in a single APDU, the .request primitive is invoked with Request_Type == NORMAL or WITH-LIST as appropriate. Otherwise, it is invoked with Request_Type == FIRST-BLOCK or WITH-LIST-AND-FIRST-BLOCK as appropriate, then with Request_Type == ONE-BLOCK and finally with LAST-BLOCK. Upon 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 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 reception of an Action-Response APDU.

Once all method invocation parameters are transferred, the server invokes the methods of COSEM interface 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 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 7.3.5.

6.9 The EventNotification service

Function

The EventNotification service is an unsolicited, non-client/server type service. It is requested by the server, upon 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 19.

Table 19 – 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, those 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 interface 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 interface object.

Use

A possible logical sequence of the `EventNotification` service primitives is illustrated in Figure 8 f) and g).

The `.request` primitive is invoked by the server AP to send the value of a COSEM interface object attribute to the remote client AP. Upon reception of the `.request` primitive, the Server AL builds the `EventNotificationRequestAPDU`.

In some cases, the supporting lower 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 `Trigger_EventNotification_Sending` service primitive.

The `EventNotification.indication` primitive is generated by the client AL upon reception of an `EventNotificationRequestAPDU`.

The protocol for the `EventNotification` service is specified in 7.3.6.

6.10 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.

NOTE This service is used 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 20.

Table 20 – Service parameters of the `TriggerEventNotificationSending.request` service primitive

	.request
<code>Protocol_Parameters</code>	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 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.

6.11 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 21:

- 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 21 – 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		

6.12 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 interface 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 interface 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 22.

Table 22 – 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 Table 12.				

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 23.

Table 23 – 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	
	References a list ¹ of COSEM object methods, with method invocation parameters.	Data {Data}	Delivers the method invocation return parameters. NOTE 1 If parameters are returned, this implies that the method invocation succeeded.
		Data_Access_Error {Data_Access_Error}	Provides the reason for the method invocation to fail.
		Data_Block_Result	As above.
Read_Data_Block_Access	Carries block transfer control information and one part of encoded form of the COSEM method references and method invocation parameters.	Block_Number	Carries the number of the latest data block received.
Block_Number_Access	Carries the number of the latest data block received.	Data_Block_Result	As above.
NOTE 2 The same Read.response choice can 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;

NOTE 1 With this, the Read service can transport information in both directions, just like the ACTION service used with LN referencing: method invocation parameters from the client to the server and return parameters from the server to the client.

- the Read_Data_Block_Access variant is used when one or more COSEM object methods are invoked and the encoded form of the request does not fit in a single APDU. The request may include a single Read_Data_Block_Access parameter. It carries block transfer control information and raw data:
 - the Last_Block element indicates if the given block is the last one (TRUE) or not (FALSE);

- the `Block_Number` element carries the number of the actual block sent;
- the `Raw_Data` element carries a part of the encoded form of the list of `Variable_Access_Specification` parameters (as it would be used without block transfer) including the method references and the method invocation parameters. Here, only the variants `Variable_Name` and `Parameterized_Access` are allowed.
- the `Block_Number_Access` variant is used when the server uses block transfer to send a long response, to confirm the reception of a data block and to request the next data block. The request may include a single `Block_Number_Access` parameter. It carries the number of the latest data block received correctly.

The Result (+) parameter indicates that the requested service has succeeded.

Without block transfer, the `.response` / `.confirm` service primitives contain one or more `Read_Result` parameters. Their number and order shall be the same as that of the `Variable_Name` / `Parameterized_Access` parameters in the `.request` / `.indication` primitives.

If the Read service is used to read attribute(s), then:

- the Data choice is taken to carry the value of the attribute at the time of access;
- the `Data_Access_Error` is taken to carry the reason for the read to fail for this attribute.

If the Read service is used to invoke method(s), then:

- the Data choice is taken to carry the return parameters (if data are returned, this implies that the method invocation succeeded). If there are no return parameters, Data shall be null data;

NOTE 2 However, if no return data are expected, the Write service is 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 shall carry 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 8 item a).

The Read.request primitive is invoked following the invocation of a GET or ACTION .request primitive by the client AP and mapping this to a Read.request primitive by the SN_MAPPER ASE. The client AL builds then the ReadRequest APDU and sends it to the server. For LN / SN service mapping, see 7.3.7.

The Read.indication primitive is generated by the server AL upon 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 then mapped back to a GET or ACTION .confirm primitive by the SN_MAPPER ASE and the GET or ACTION .confirm primitive is generated.

The protocol of the Read service is specified in 7.3.7.

6.13 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 interface object attributes;
- to invoke one or more COSEM interface 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 24.

Table 24 – Service parameters of the Write service

	.request	.indication	.response	.confirm
Variable_Access_Specification { Variable_Access_Specification }	M	M(=)		
Variable_Name	S	S (=)		
Parameterized_Access	S	S (=)	–	–
Variable_Name	M	M (=)		
Selector	M	M (=)		
Parameter	M	M (=)		
Write_Data_Block_Access	S	S (=)		
Last_Block	M	M (=)	–	–
Block_Number	M	M (=)		
Data { Data }	M	M (=)	–	–
Result (+)	–	–	S	S (=)
Write_Result { Write_Result }	–	–	S	S (=)
Successss			S	S (=)
Data_Access_Error			S	S (=)
Block_Number			S	S (=)
Result (–)			S	S (=)
Error_Type			M	M (=)
NOTE For security parameters, see Table 12.				

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 25. The use of the Data service parameter is also explained.

Table 25 – 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.
		Data_Access_Error {Data_Access_Error}	Provides the reason for the write to fail.
Parameterized_Access {Parameterized_Access}	References a list ¹ of COSEM object attributes to be written selectively. The Data service parameter carries the data to be written.	Success {Success}	As above.
		Data_Access_Error {Data_Access_Error}	
Write_Data_Block_Access	Carries block transfer control information. The Data service parameter carries raw-data, including the encoded form of the list ¹ of COSEM object attribute or method references, and the list of data to be written or the list of method invocation parameters.	Block_Number	Carries the number of the latest data block received.
NOTE The same Write.response choice can 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 8 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 SN_MAPPER ASE. The client AL builds then the WriteRequest APDU and sends it to the server. For LN / SN service mapping, see 7.3.8.

The Write.indication primitive is generated by the server AL upon 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 SN_MAPPER ASE and the SET or ACTION .confirm primitive is generated.

The protocol of the Write service is specified in 7.3.8.

6.14 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 interface object method when no return parameters are expected.

The UnconfirmedWrite request service primitive shall always fit in a single APDU.

The UnconfirmedWrite service is specified in IEC 61334-4-41:1996, 10.6 and Annex A. For completeness and for consistency with the specification of services using LN referencing, the specification is reproduced here, together with the extensions made for DLMS/COSEM.

Semantics

The UnconfirmedWrite service primitives shall provide service parameters as shown in Table 26.

Table 26 – 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 12.		

The use of the different variants of the Variable-Access-Specification service parameter of the UnconfirmedWrite.request service primitive is shown in Table 27. The use of the Data service parameter is also explained.

Table 27 – 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 8 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 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 reception of a WriteRequest APDU.

The protocol of the UnconfirmedWrite service is specified in 7.3.9.

6.15 The InformationReport service

Function

The InformationReport service is an unsolicited, non-client/server type service. It is requested by a server using SN referencing, upon occurrence of an event, in order to inform the client of the value of one or more DLMS named variables – mapped to COSEM interface 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 28.

Table 28 – 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 interface 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 7.3.10.

6.16 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 (see Table 29):

Table 29 – 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 IEC 62056-6-2.

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.

6.17 Summary of services and LN/SN data transfer service mapping

The following Tables 30, 31 and 32 provide a summary of the COSEM application layer services.

Table 30 – 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 31 – Summary of xDLMS services for LN referencing

Client side	Server side
GET.request	GET.indication
GET.confirm	GET.response
SET.request	SET.indication
SET.confirm	SET.response
ACTION.request	ACTION.indication
ACTION.confirm	ACTION.response
EventNotification.indication	EventNotification.request
Trigger_EventNotification_Sending.request	–

Table 32 – Summary of xDLMS services for SN referencing

Client side	Server side
Read.request	Read.indication
Read.confirm	Read.response
Write.request	Write.indication
Write.confirm	Write.response
UnconfirmedWrite.request	UnconfirmedWrite.indication
InformationReport.indication	InformationReport.request

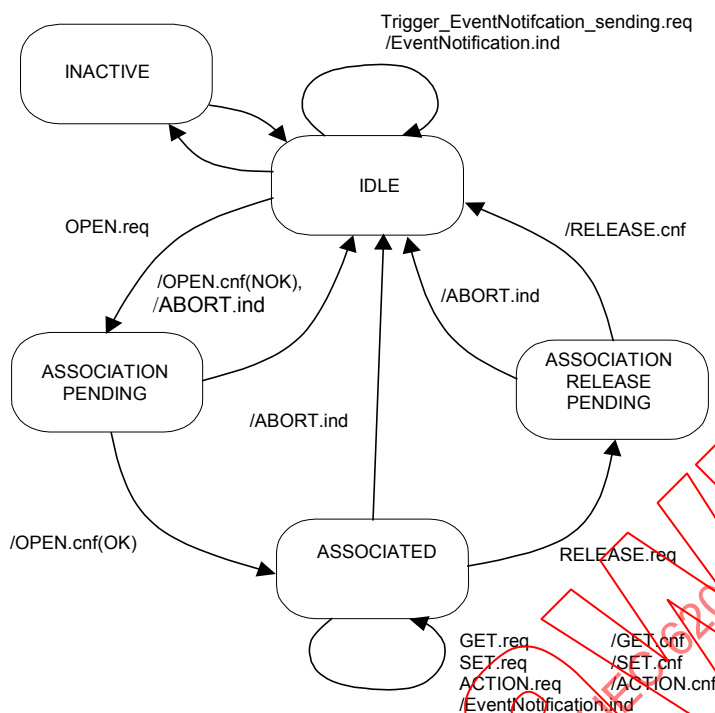
When the server uses SN referencing, a mapping between the service primitives using LN referencing and SN referencing takes place on the client side. This mapping is specified in 7.3.7, 7.3.8, 7.3.9 and 7.3.10.

7 COSEM application layer protocol specification

7.1 The control function

7.1.1 State definitions of the client side control function

Figure 9 shows the state machine for the client side CF; see also Figure 1.



IEC 1120/13

NOTE 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 9 – 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 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 happens 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, (/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. The client/server type data services are available in this state, and the EventNotification service can be handled. 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

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.

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).

7.1.2 State definitions of the server side control function

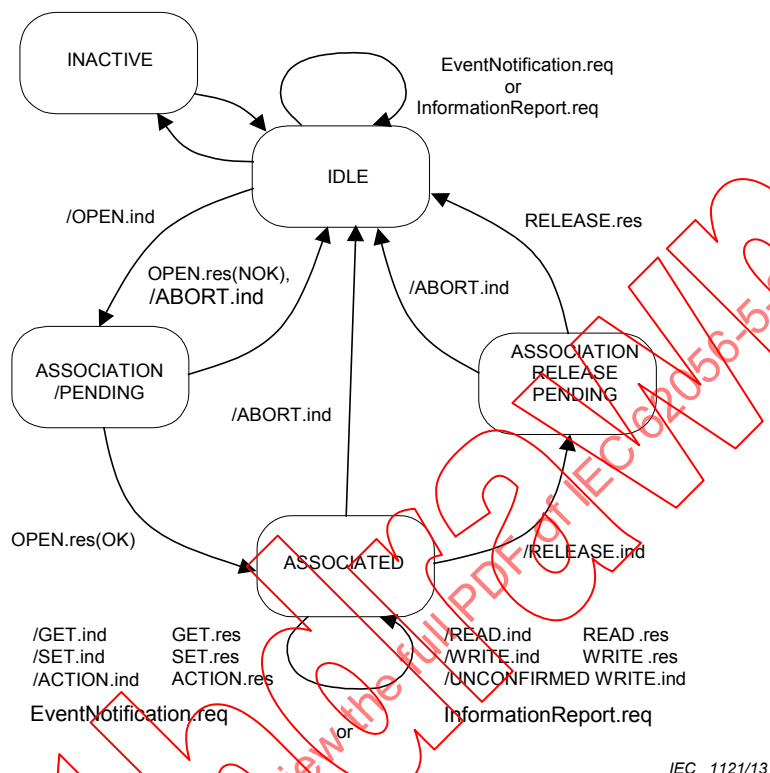


Figure 10 – Partial state machine for the server side control function

Figure 10 shows the state machine for the server side CF; see also Figure 1.

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. The client/server type data services are available in this state, and the EventNotification / InformationReport services can be handled. 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 requests 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

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.

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).

7.2 The ACSE services and APDUs

7.2.1 ACSE functional units, services and service parameters

The COSEM AL ACSE is based on the connection-oriented ACSE, as specified in ISO/IEC 15953 and ISO/IEC 15954.

Functional units are used to negotiate ACSE user requirements during association establishment. Three functional units are defined:

- Kernel functional unit;
- Authentication functional unit; and
- Application Context Negotiation functional unit.

The COSEM AL uses only the Kernel and the Authentication functional units.

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. It is the default functional unit. To be included, the Authentication functional unit shall be explicitly requested on the AARQ APDU and accepted on the AARE APDU.

The selection of the Authentication functional unit supports additional parameters on the AARQ and AARE APDUs. The Authentication functional unit does not affect the elements of procedure. Table 33 shows the services and parameters associated with the ACSE functional units, as used by the COSEM AL. The abstract syntax of the ACSE APDUs is specified in Clause 8.

Table 33 – ACSE functional units, services and service parameters

Functional Unit	Service	APDU	parameter	Presence
Kernel	A-ASSOCIATE	AARQ	protocol-version	O ¹⁾
			application-context-name	M
			called-AP-title	U
			called-AE-qualifier	U
			called-AP-invocation-identifier	U
			called-AE-invocation-identifier	U
			calling-AP-title	U
			calling-AE-qualifier	U
			calling-AP-invocation-identifier	U
			calling-AE-invocation-identifier	U
			implementation-information	O
			user-information ²⁾	M
			(carrying an xDLMS-Initiate.request APDU)	
			dedicated-key	U
			response-allowed	U
			proposed-quality-of-service	U
			proposed-dlms-version-number	M
			proposed-conformance	M
			client-max-receive-pdu-size	M
		AARE	protocol-version	O
			application-context-name	M
			result	M
			result-source-diagnostic	M
			responding-AP-title	U
			responding-AE-qualifier	U
			responding-AP-invocation-identifier	U
			responding-AE-invocation-identifier	U
			implementation-information	O
			user-information ²⁾	M
			(carrying an xDLMS-initiate.response APDU)	S
			negotiated-quality-of-service	U
			negotiated-dlms-version-number	M
			negotiated-conformance	M
			server-max-receive-pdu-size	M
			vaa-name	M
			(or carrying a ConfirmedServiceError APDU)	S
	A-RELEASE	RLRQ	reason	U
			user-information	U
		RLRE	reason	U
			user-information	U
Authentication	A-ASSOCIATE	AARQ	sender-acse-requirements	U
			mechanism-name	U
			calling-authentication-value	U

Functional Unit	Service	APDU	parameter	Presence
		AARE	responder-acse-requirements	U
			mechanism-name	U
			responding-authentication-value	U
¹⁾ O: Presence is an ACPM (COSEM Application layer) option. ²⁾ According to ISO/IEC 15953, the user-information parameter is optional. However, in the COSEM environment it is mandatory in the AARQ / AARE APDUs.				

In general, the value of each parameter of the AARQ APDU is determined by the parameters of the COSEM-OPEN.request service primitive. Similarly, the value of each parameter of the AARE is determined by the COSEM-OPEN.response primitive. The COSEM-OPEN service is specified in 6.2.

The AARQ and AARE APDU parameters are specified below. Managing these parameters is specified in 7.2.4.1.

- **protocol-version**: the COSEM AL uses the default value. For details, see ISO/IEC 15954;
- **application-context-name**: COSEM application context names are specified in 7.2.2.2;
- **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;
- **implementation-information**: this parameter is not used by the COSEM AL. For details, see ISO/IEC 15954;
- **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 parameter 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 module selected;
- **mechanism-name**: COSEM authentication mechanism names are specified in 7.2.2.3;
- **calling- and responding- authentication-value**: see 5.3;
- **result**: the value of this parameter is determined by the COSEM AP (acceptor) or the 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 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 parameter 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 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 6.3) is invoked with the parameter Use_RLRQ_RLRE == TRUE – are specified below.

- **reason:** carries the appropriate value as specified in 6.3;
- **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 6.3.

7.2.2 Registered COSEM names

7.2.2.1 General

Within an OSI environment, many different types of network objects shall 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 User Association, 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-organisation(5) DLMS-UA(8) }
```

For DLMS/COSEM, object identifiers are specified for naming the following items:

- COSEM application context names;
- COSEM authentication mechanism names.

7.2.2.2 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 establishment¹⁰. 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-organisation(5) DLMS-UA(8) application-  
context(1) context_id(x)}
```

The meaning of these general COSEM application contexts is:

- there are two ASEs present within the AE invocation, the ACSE and the xDLMS_ASE;

¹⁰ 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 xDLMS_ASE shall be as it is specified in IEC 61334-4-41¹¹;
- the transfer syntax shall be A-XDR, specified in IEC 61334-6.

The specific context_ids and the use of ciphered and unciphered APDUs is shown in Table 34 below:

Table 34 – Use of ciphered / unciphered APDUs

Application context name	context_id	Unciphered APDUs	Ciphered APDUs
Logical_Name_Referencing_No_Ciphering::=	context_id(1)	Yes	No
Short_Name_Referencing_No_Ciphering::=	context_id(2)	Yes	No
Logical_Name_Referencing_With_Ciphering::=	context_id(3)	Yes	Yes
Short_Name_Referencing_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.

7.2.2.3 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 authentication (lowest level) security;
- low level, password based authentication security (LLS) identifying only the client;
- high level, four-pass authentication security (HLS) identifying both the client and the server.

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:

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)
NOTE With mechanism_id(2), the method of processing the challenge is secret.	

¹¹ With the COSEM extensions to DLMS; see 4.2.3.1.

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 5.3 and 5.4.8.4.

7.2.3 APDU encoding rules

The ACSE APDUs shall be encoded in BER, in accordance with ISO/IEC 8825-1. The user-information field of these APDUs shall carry the xDLMS `InitiateRequest` / `InitiateResponse` / `ConfirmedServiceError` APDU as appropriate, encoded in A-XDR. The resulting OCTET STRING shall be encoded in BER.

Examples for AARQ/AARE APDU encoding are given in Annex B and in Annex C.

7.2.4 Protocol for application association establishment

7.2.4.1 Protocol for the establishment of confirmed application associations

AA establishment using the A-Associate service of the ACSE is the key element of COSEM interoperability. The participants of an AA are:

- a client AP, proposing AAs; and
- a server AP ¹², accepting them or not.

Figure 11 gives the MSC for the case, when:

- the COSEM-OPEN.request primitive requests a confirmed AA;
- the connection of the supporting lower 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`. 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 ¹³. 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 of the APDU ¹⁴ received and enters the ASSOCIATION PENDING state.

NOTE 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.

¹² To support multicast and broadcast services, an AA can also be established between a client AP and a group of server APs.

¹³ The PH layer shall be connected before the COSEM-OPEN service is invoked.

¹⁴ 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 Annex A.

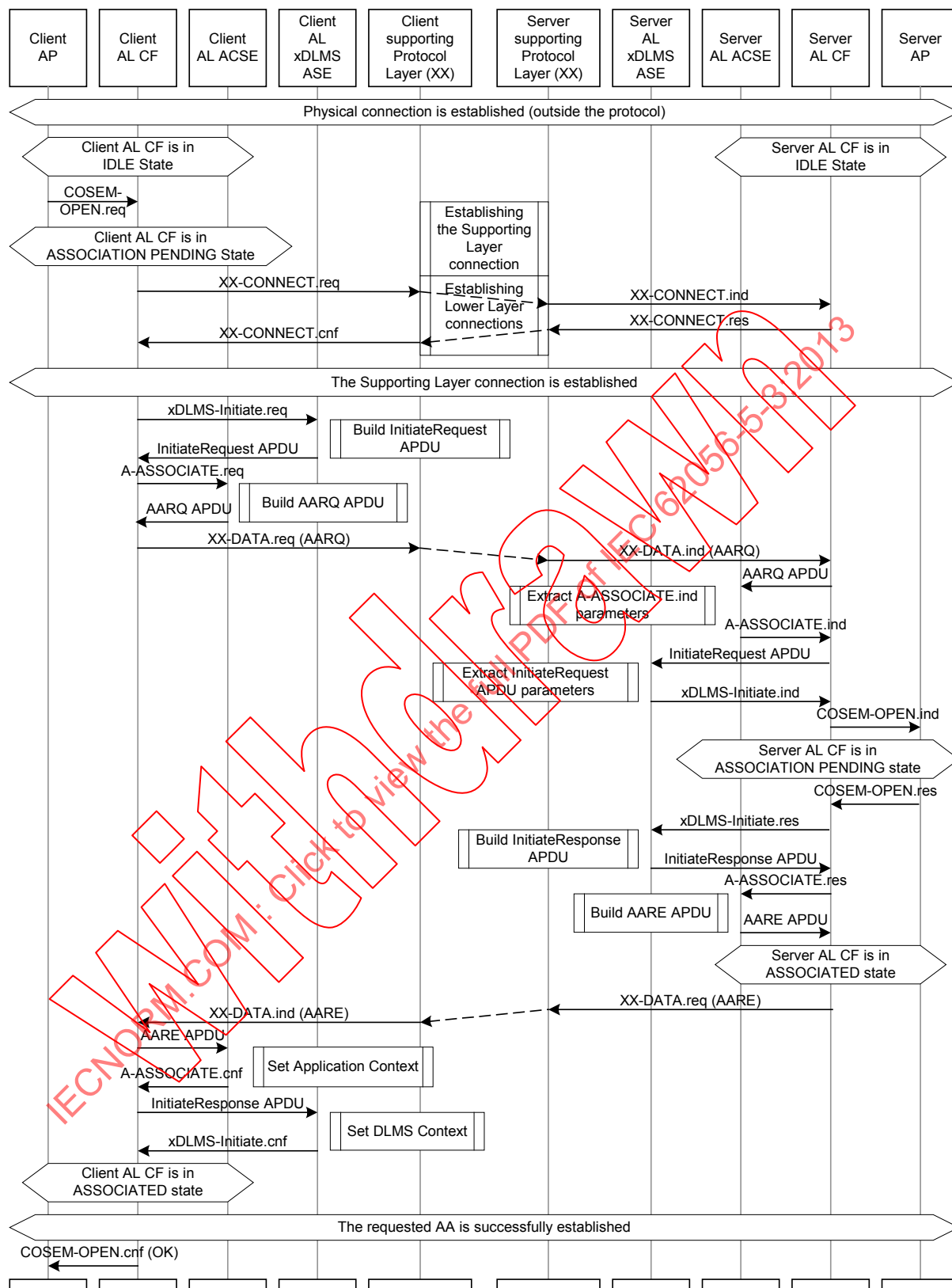


Figure 11 – MSC for successful AA establishment preceded by a successful lower layer connection establishment

IEC 1122/13

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 shall carry the same system title. Otherwise, the AA shall be rejected and appropriate diagnostic information shall be sent;

Fields of the authentication functional unit (when present):

- sender-acse-requirements:
- if is not present or it is present but bit 0 = 0, then the authentication functional unit is not selected. Any following fields of the authentication functional unit may be ignored;
- 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, the value of this parameter is TRUE (default), the server shall send back an AARE APDU. Otherwise, the server shall not respond. See also Annex A.
 - a) proposed-dlms-version-number. See 4.2.3.1;
 - b) proposed-conformance. See 7.3.1;
 - c) client-max-receive-pdu-size. See 4.2.3.1.

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;
- 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 shall carry the same system title. Otherwise, the AA shall be aborted by the client.
- Result: accepted;
- Failure_Type: Result-source: acse-service-user; Diagnostic: null;
- User_Information: an xDLMS InitiateResponse APDU carrying the elements of the 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 lower 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 parameters of the AARE APDU received are extracted with the help of the ACSE and the xDLMS ASE, and passed to the client AP via the COSEM-OPEN.confirm

service primitive. At the same time, the client AL enters the ASSOCIATED state. The AA is established now with the application context and xDLMS context negotiated.

If the application context proposed by the client is not acceptable or the authentication of the client is not successful, the COSEM-OPEN.response primitive is invoked with the following parameters:

- Application_Context_Name: the same as the one proposed, or the one supported by the server;
- Result: rejected-permanent or rejected-transient;
- Failure_Type: Result-source: acse-service-user; Diagnostic: an appropriate value;
- User_Information: an xDLMS InitiateResponse APDU with the parameters of the xDLMS context supported by the server.

If the application context proposed by the client is acceptable and the authentication of the client is successful but the xDLMS context cannot be accepted, the COSEM-OPEN.response primitive shall be invoked with the following parameters:

- Application_Context_Name: the same as the one proposed;
- Result: rejected-permanent or rejected-transient;
- Failure_Type: Result-source: acse-service-user; Diagnostic: no-reason-given;
- User_Information: an xDLMS ConfirmedServiceError APDU, indicating the reason for not accepting the proposed xDLMS context.

In these two cases, upon invocation of the .response primitive, the CF assembles and sends the AARE APDU to the client via the supporting lower layer protocols. The proposed AA is not established, the server CF returns to the IDLE state.

At the client side, the parameters 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 situation, it returns a COSEM-OPEN.response primitive to the client with an appropriate Result parameter. The Result Source parameter is appropriately assigned the symbolic value of "ACSE service-provider". The COSEM-OPEN.indication primitive is not issued. The association is not established.

7.2.4.2 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 7.2.4.4.

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 ExceptionResponse APDU.

7.2.4.3 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.

7.2.4.4 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 ¹⁵.

This standard 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.

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.

7.2.5 Protocol for application association release

7.2.5.1 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.

7.2.5.2 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

¹⁵ As for all AAs, the logical devices contain an Association LN / SN interface object for the pre-established associations, too.

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 6.3, 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 12.

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 6.3.

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 APDU 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.

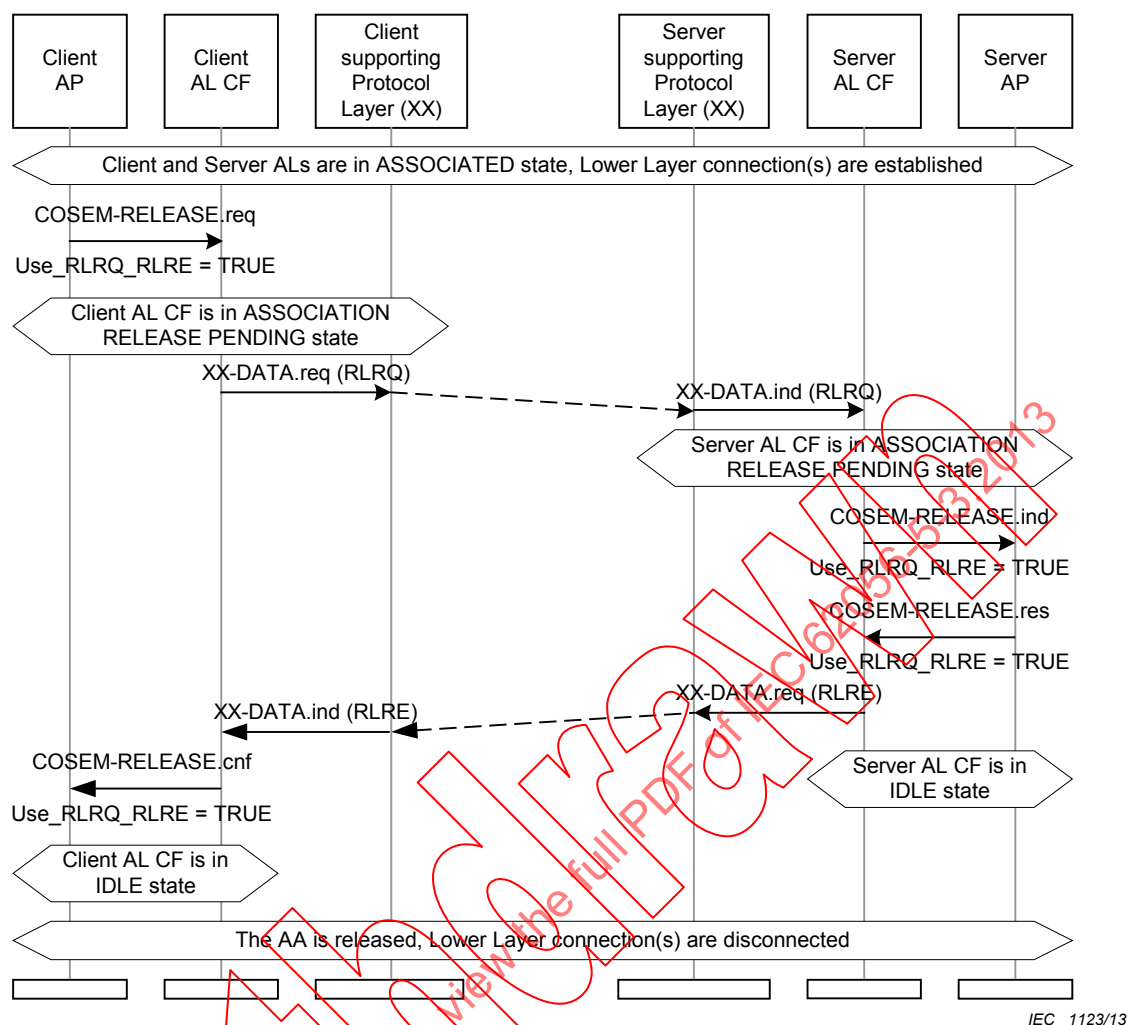


Figure 12 – Graceful AA release using the A-RELEASE service

Figure 13 gives an example of graceful releasing a confirmed AA by disconnecting the corresponding lower layer connection.

A client AP that desires to release an AA not using the A-RELEASE service, invokes the COSEM-RELEASE.request primitive with Use_RLRQ_RLRE == FALSE or not present. The client AL CF enters the ASSOCIATION RELEASE PENDING state.

In communication profiles where the RLRQ service is mandatory, invoking the .request primitive with no Use_RLRQ_RLRE or with Use_RLRQ_RLRE == FALSE may lead to an error: the .request shall be locally and negatively confirmed. The client AL CF returns to the IDLE state.

When the client AL CF receives the .request primitive, it sends an XX-DISCONNECT.request primitive to the server.

When the server AL CF receives the XX-DISCONNECT.request primitive, the CF enters the ASSOCIATION RELEASE PENDING state. The COSEM-RELEASE.indication primitive is generated by the server AL CF with Use_RLRQ_RLRE == FALSE or not present.

The COSEM-RELEASE.response primitive is invoked by the server AP to indicate if the release of the AA is accepted or not. Note, that the server AP cannot refuse a release request. Upon the reception of this primitive, the server AL CF sends an XX-DISCONNECT.response primitive to the client and returns to the IDLE state.

The COSEM-RELEASE.confirm primitive is generated by the client AL when the XX-DISCONNECT.confirm primitive is received. The supporting layer is disconnected. The client AL CF returns to the IDLE state.

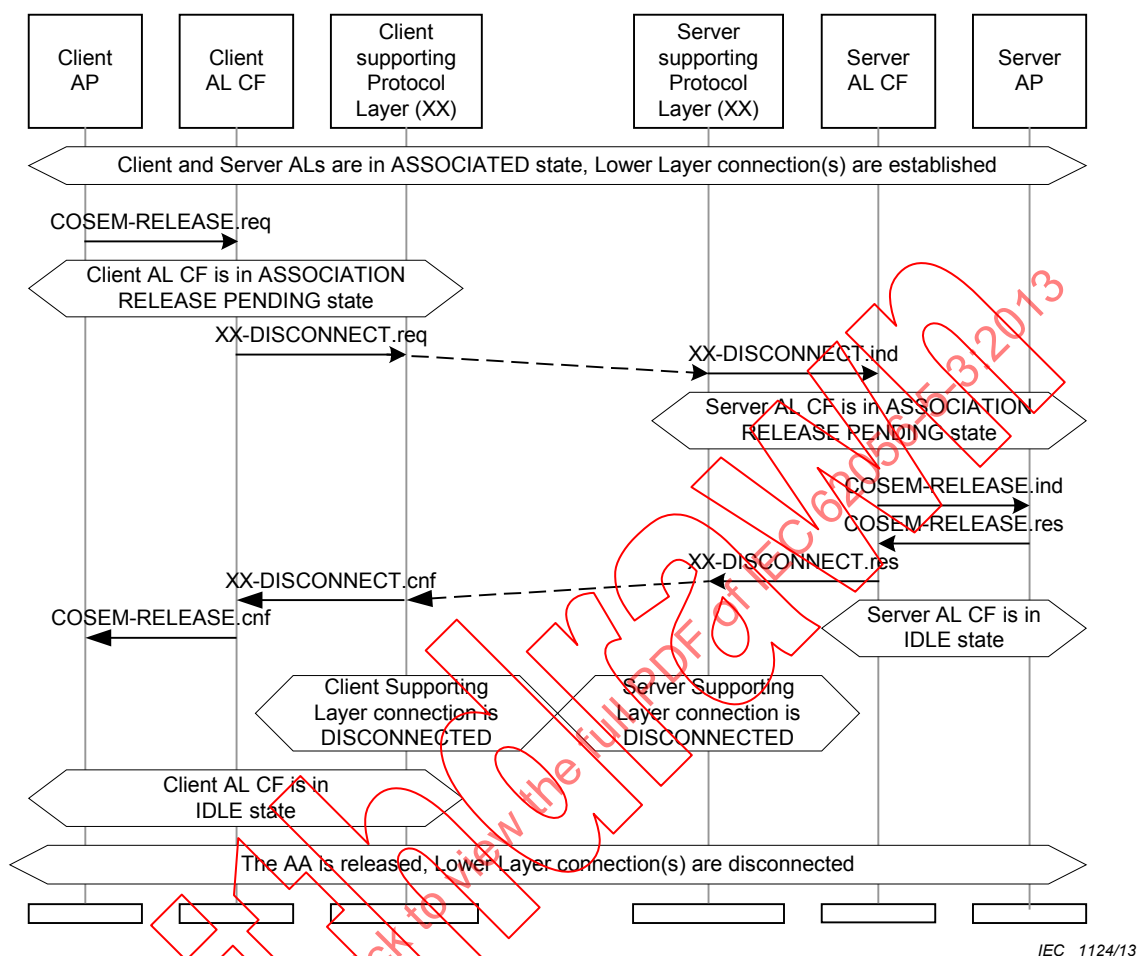


Figure 13 – Graceful AA release by disconnecting the supporting layer

7.2.5.3 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 14 shows the message sequence chart for aborting the AA, due to the detection of a physical disconnection.

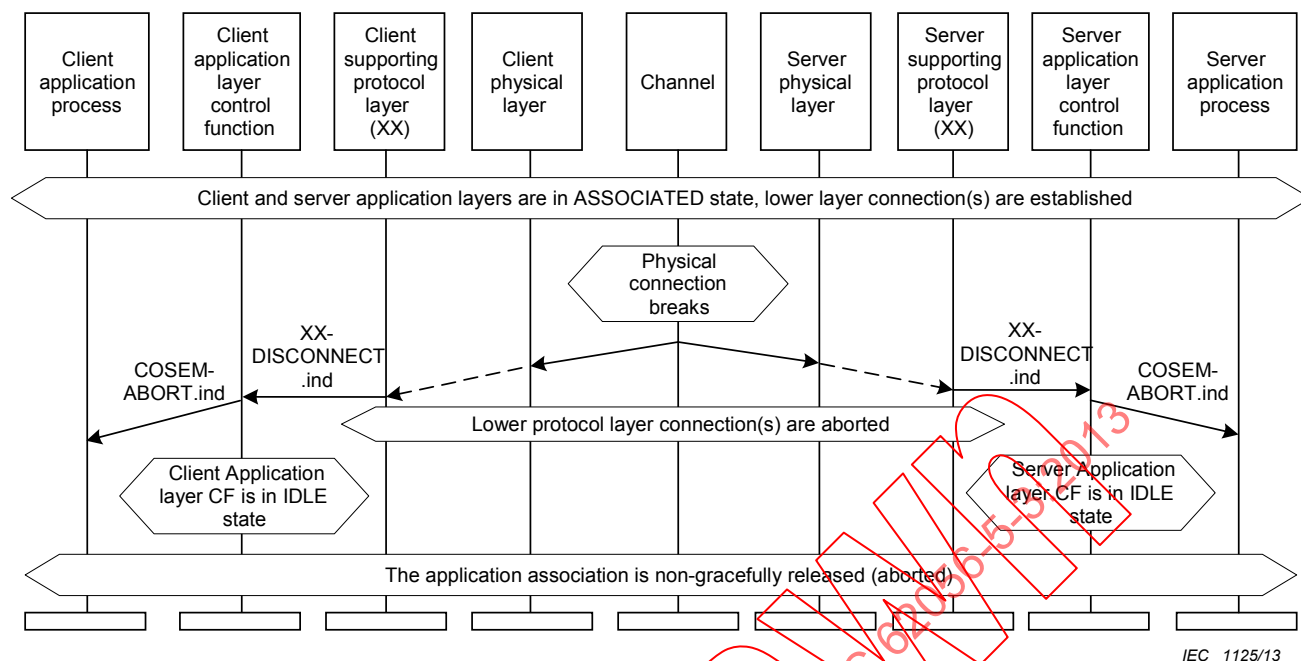


Figure 14 – Aborting an AA following a PH-ABORT.indication ¹⁶

7.3 Protocol for the data transfer services

7.3.1 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 conform to its specification. If a service or option is not present in the negotiated conformance block, it may not be requested by the client.

The xDLMS conformance block can be distinguished from the DLMS conformance block specified in IEC 61334-4-41 by its tag: APPLICATION 31. It is shown in Table 35.

¹⁶ The release of an AA may require internal communication between the ASEs of the CF. These are not shown on the figures.

Table 35 – xDLMS Conformance block

Conformance block bit	Reserved	LN referencing	SN referencing
0	x		
1	x		
2	x		
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	x		
17	x		
18			parameterized-access
19		get	
20		set	
21		selective-access	
22		event-notification	
23		action	

7.3.2 Confirmed and unconfirmed service invocations

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

- Figure 8 item a) in case of confirmed service invocations; and
- Figure 8 item d) in case of unconfirmed service invocations.

A client AP that desires to access an attribute or a method of a COSEM interface 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 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 Clause 8.

Within confirmed AAs, client/server type data transfer services can be invoked in a confirmed or unconfirmed manner.

Within unconfirmed AAs, client/server type data transfer 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).

7.3.3 Protocol for the GET service

When the client AP desires to read the value of one or more COSEM interface object attributes, it uses the GET service.

As explained in 6.6, 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, block transfer may be used. It is negotiated via bit 11 of the conformance block; see 7.3.1.

NOTE 1 The supporting layer can also provide a mechanism for transferring long messages.

The GET service primitive types and the corresponding APDUs are shown in Table 36.

Table 36 – 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 15 shows the MSC for a confirmed GET service in the case of success, without block transfer.

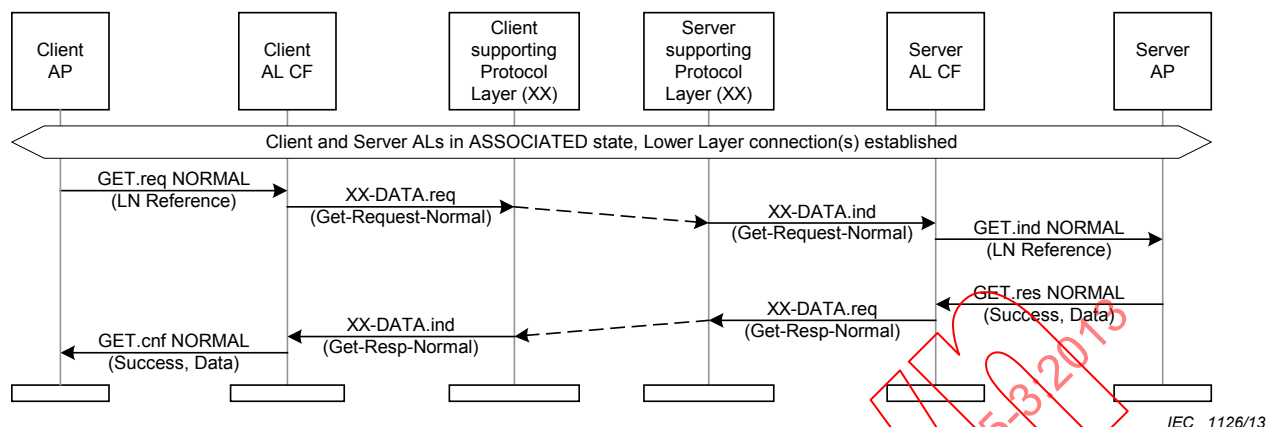


Figure 15 – MSC of the GET service

Figure 16 shows the MSC of a confirmed GET service in the case of success, with the result returned in three blocks.

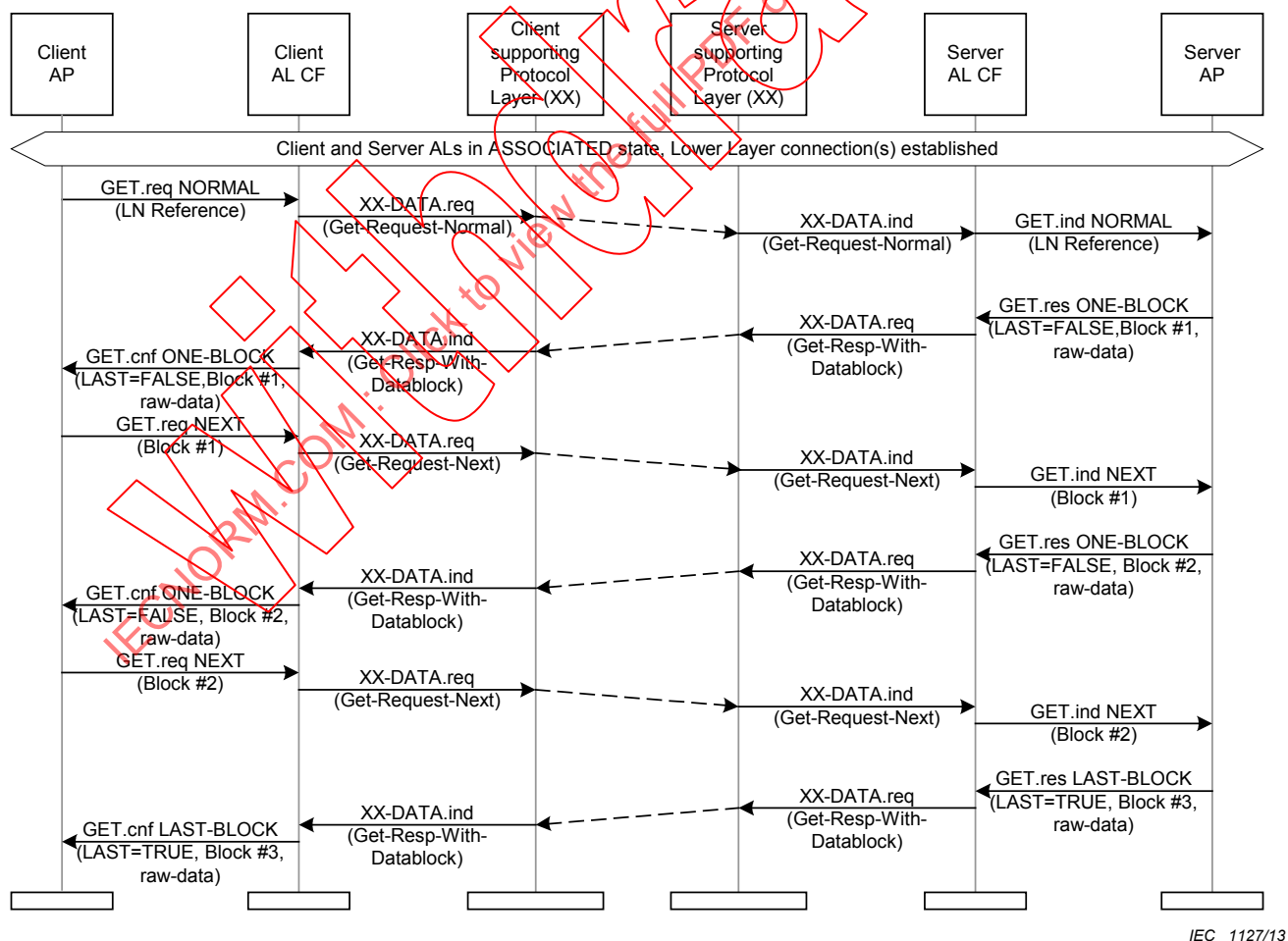


Figure 16 – MSC of the GET service with block transfer

The GET.request primitive is invoked with Request_Type == NORMAL or WITH-LIST as appropriate. As in this case the data to be returned is too long to fit in a single APDU, the server AP sends it in blocks. First, the data is encoded, as if it would fit into a single APDU:

- if the value of a single attribute was requested, only the type and value shall be encoded (Data). If the Data cannot be delivered, the response shall be of type GET-NORMAL with Data_Access_Result;
- if the value of a list of attributes was requested, then the list of results shall be encoded: for each attribute, either Data or Data_Access_Result.

The result is a series of bytes, $B_1, B_2, B_3, \dots, B_N$.

NOTE 2 The server can generate the complete response upon 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 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.

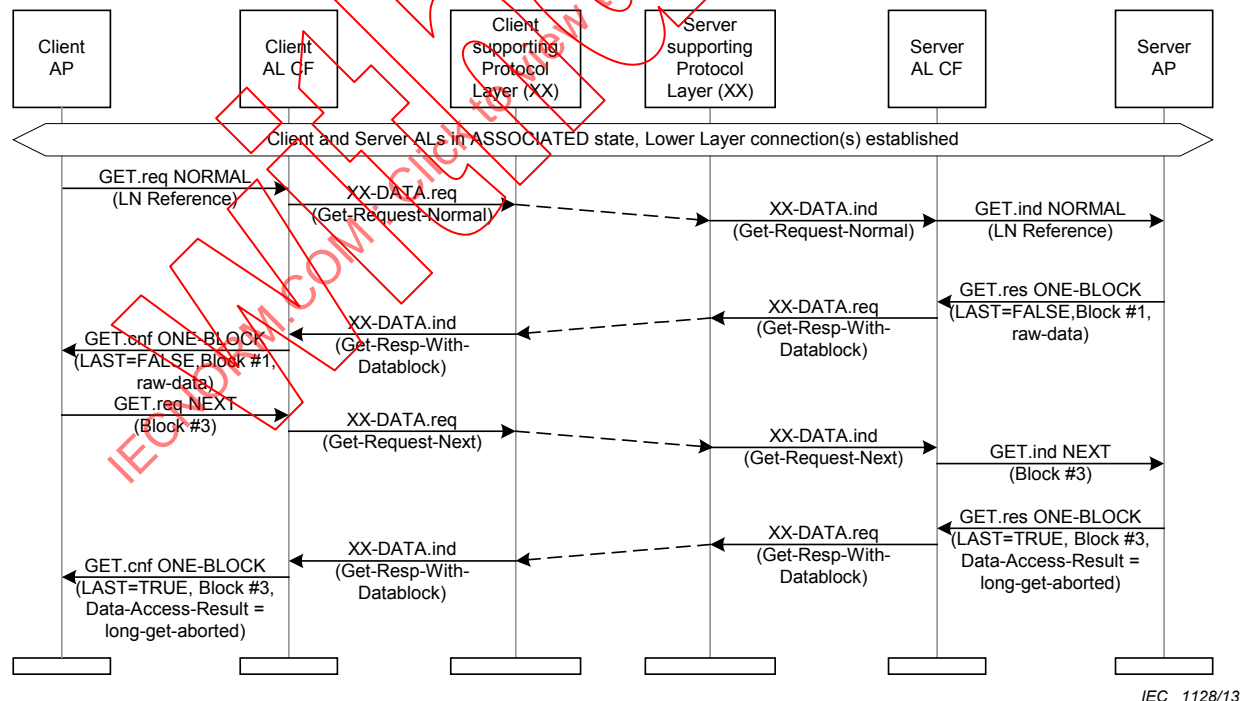
c) The server may receive a Get-Request-Next APDU when no long data transfer is in progress. In this case, the server AP shall invoke a GET-RESPONSE-LAST-BLOCK service primitive. The Result parameter shall contain a DataBlock_G structure with:

- 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 17:



IEC 1128/13

Figure 17 – MSC of the GET service with block transfer, long GET aborted

7.3.4 Protocol for the SET service

When the client AP desires to write the value of one or more COSEM interface object attributes, it uses the SET service.

As explained in 6.7, the encoded form of the request may fit in a single request or not. In this latter case, block transfer may be used. It is negotiated via bit 12 of the conformance block; see 7.3.1.

NOTE 1 The supporting layer can also provide a mechanism for transferring long messages.

The SET service primitive types and the corresponding APDUs are shown in Table 37.

Table 37 – 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
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 18 shows the MSC of a confirmed SET service, in the case of success, without block

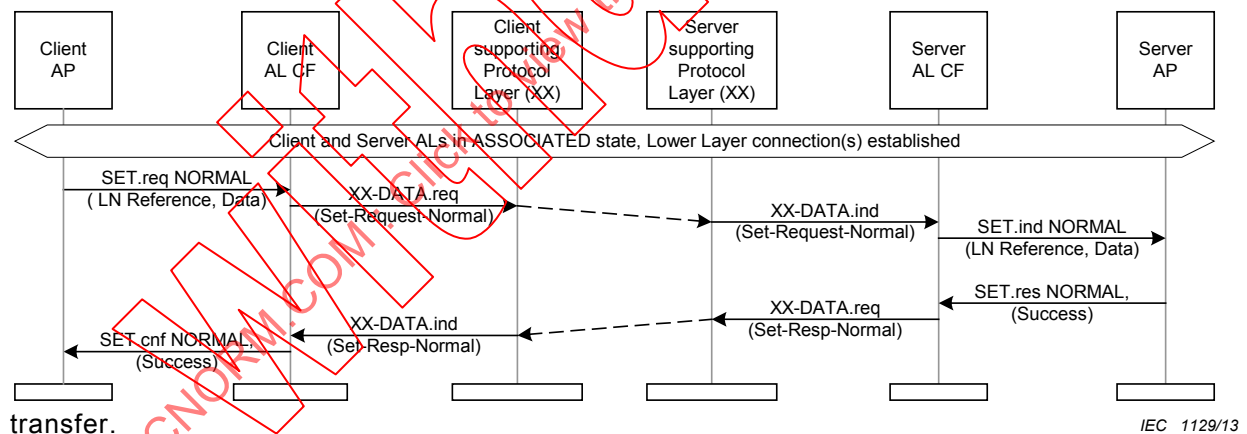


Figure 18 – MSC of the SET service

Figure 19 shows the MSC of a confirmed SET service in the case of success, with the request sent in three blocks.

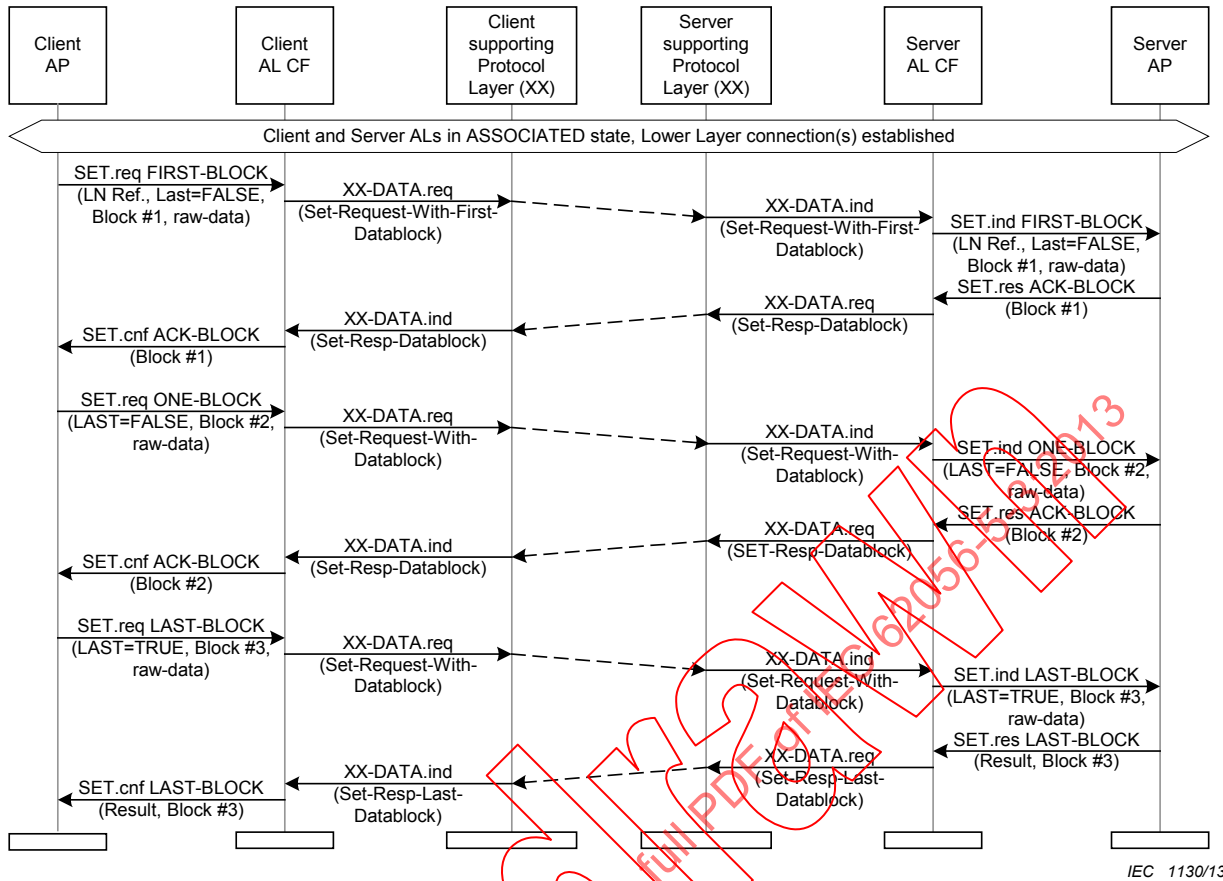


Figure 19 – MSC of the SET service with block transfer

As the data to be sent is too long to fit in a single APDU in this case, 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, $B_1, B_2, B_3, \dots, B_N$.

NOTE 2 The client can 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 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.

7.3.5 Protocol for the ACTION service

When the client AP desires to invoke one or more COSEM interface objects methods, it uses the ACTION service. As explained in 6.8, 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, block transfer may be used from the client to the server and/or from the server to the client respectively. It is negotiated via bit 13 of the conformance block; see 7.3.1.

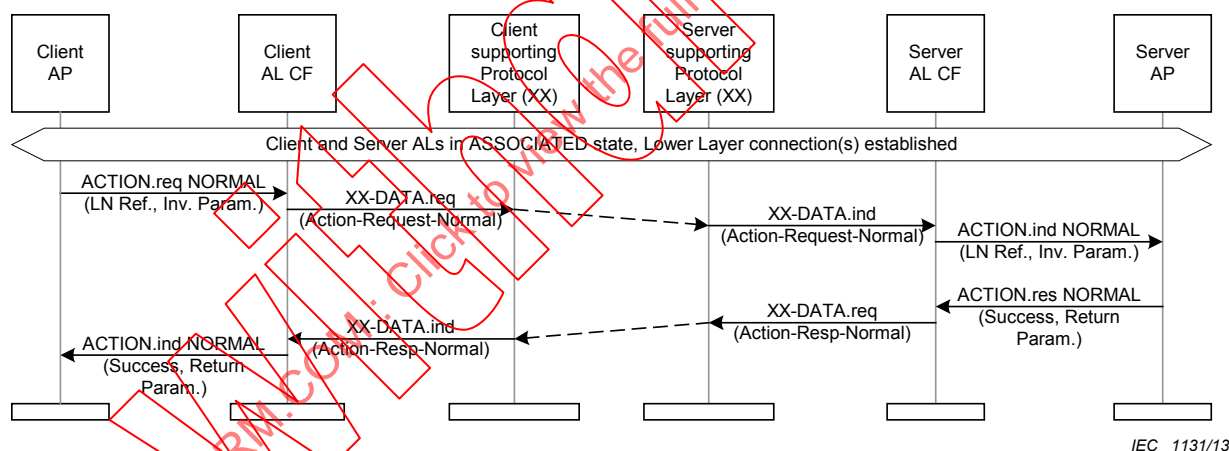
NOTE The supporting layer can also provide a mechanism for transferring long messages.

The ACTION service primitive types and the corresponding APDUs are shown in Table 38.

Table 38 – ACTION service types and APDUs

ACTION .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-With-First-Pblock	Action-Response-Next-Pblock	NEXT

Figure 20 illustrates the MSC of a confirmed ACTION service in the case of success, without block transfer.



IEC 1131/13

Figure 20 – 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 21 illustrates the MSC in the case when block transfer takes place in both directions.

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.

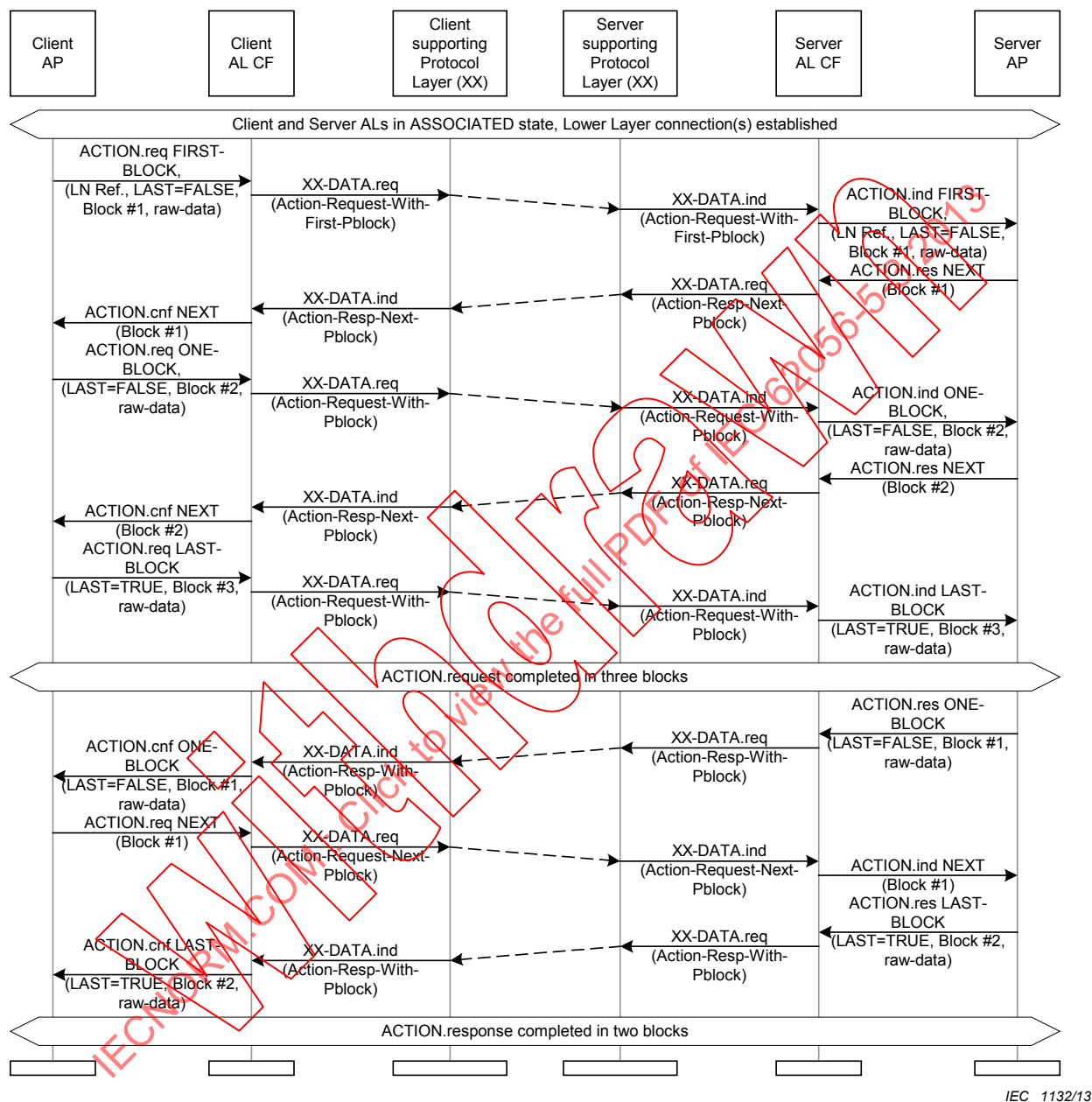


Figure 21 – MSC of the ACTION service with block transfer

7.3.6 Protocol for the EventNotification service

Upon invocation of the EventNotification.request service, the Server AL builds an EventNotificationRequest 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 the parts of IEC 62056 describing the communication profiles;

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 invocation of this primitive, the server AL builds an EventNotificationRequest 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 reception of the EventNotificationRequest APDU, the client AL generates an EventNotification.indication primitive¹⁷ to the COSEM client AP;
- by default, event notifications are sent from the management logical device (server) to the management AP (client).

7.3.7 Protocol for the Read service

As explained in 0, the Read service is used when the server uses SN referencing, either to read (a) COSEM interface 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 are shown in Table 39;
- 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 40.

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 39 – Mapping between the GET and the Read services

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)

¹⁷ At the client side, it is always EventNotification.indication, independently of the referencing scheme (LN or SN) used by the server.

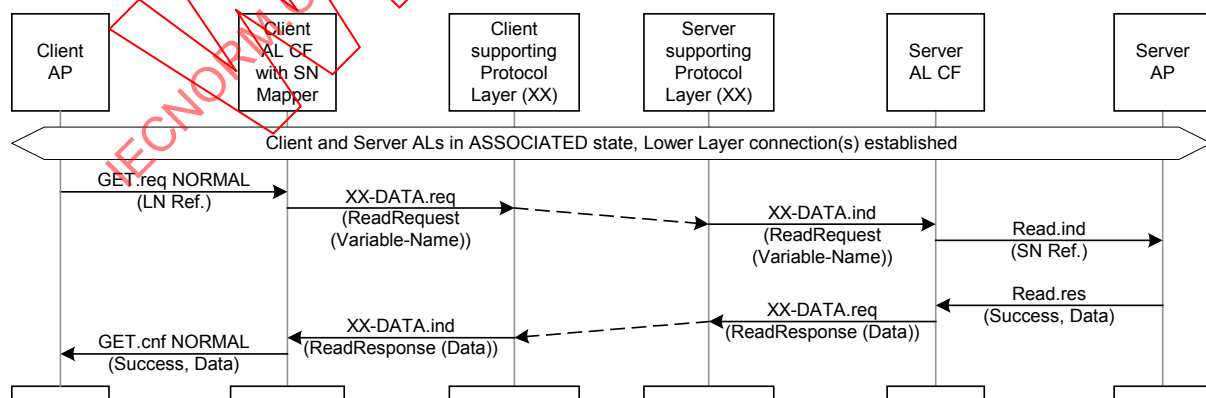
From GET.request of type	To Read.request	SN APDU
ONE-BLOCK	ReadResponse::= (data-block-result)	Read.response (Data_Block_Result) with Last_Block = FALSE
LAST-BLOCK	ReadResponse::= (data-block-result)	Read.response (Data_Block_Result) with Last_Block = TRUE
WITH-LIST	ReadResponse::= ({data data-access-error})	Read.response ({Data Data_Access_Error})

Table 40 – Mapping between the ACTION and the Read services

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)
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)

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



IEC 1133/13

Figure 22 – MSC of the Read service used for reading an attribute

Figure 23 shows the MSC of a Read service used to invoke a single method.

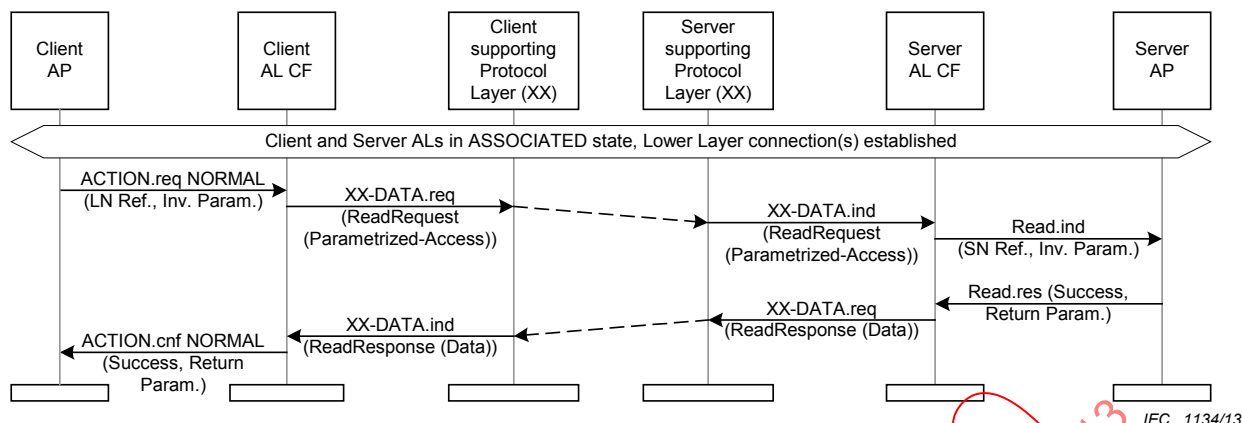


Figure 23 – MSC of the Read service used for invoking a method

Figure 24 shows the MSC of a Read service for reading a single attribute, with the result returned in three blocks.

The process of preparing and transporting the long data is essentially the same as in the case of the GET and ACTION services.

- 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.

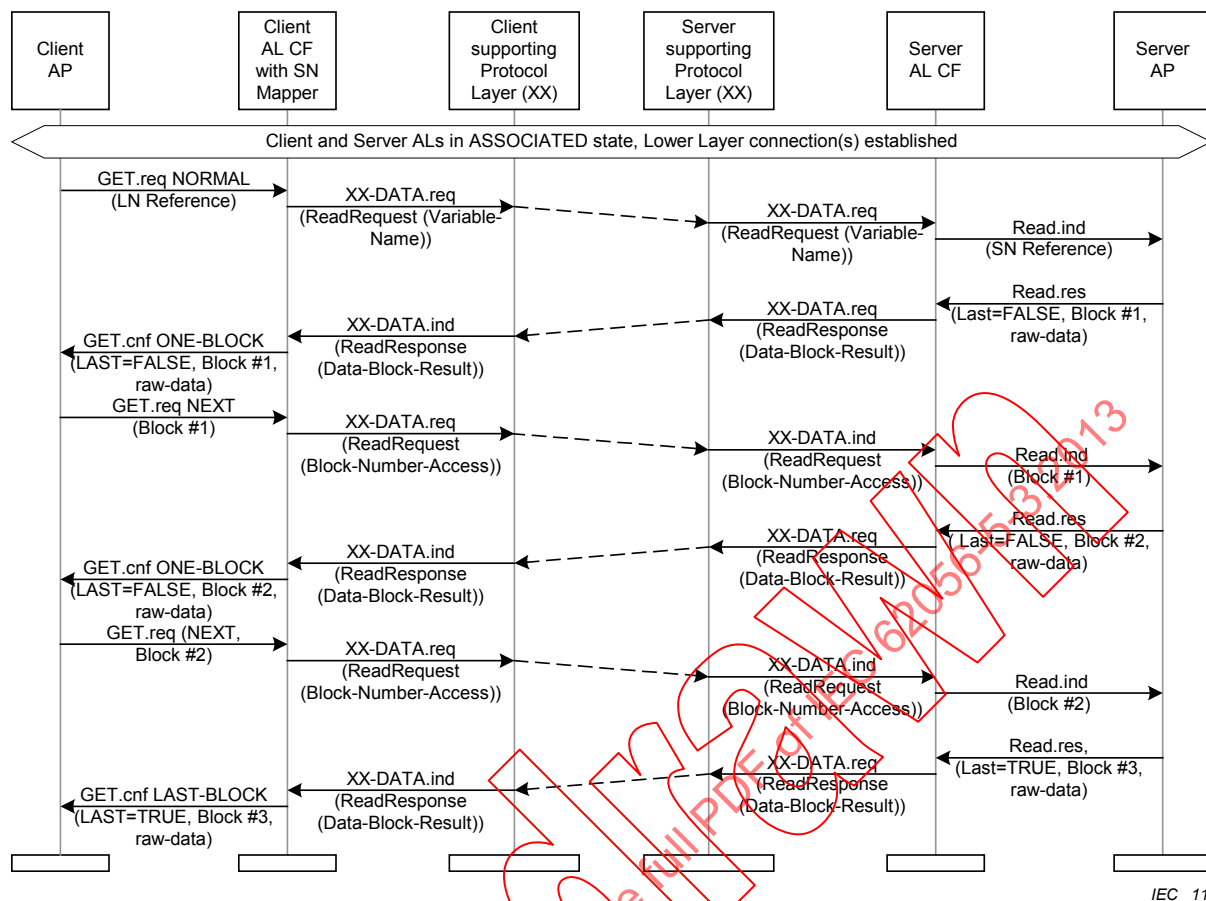


Figure 24 – MSC of the Read Service used for reading an attribute, with block transfer

7.3.8 Protocol for the Write service

As explained in 6.13, 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 41;
- 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 42.

Table 41 – Mapping between the SET and the Write services

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 = Write_Data_Block_Access; with Last_Block = FALSE, Block_Number = 1 Data = as above	WriteRequest::= (write-data-block-access, data)
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 42 – 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)
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 25 shows the MSC of a Write service used to write the value of a single attribute, in the case of success.

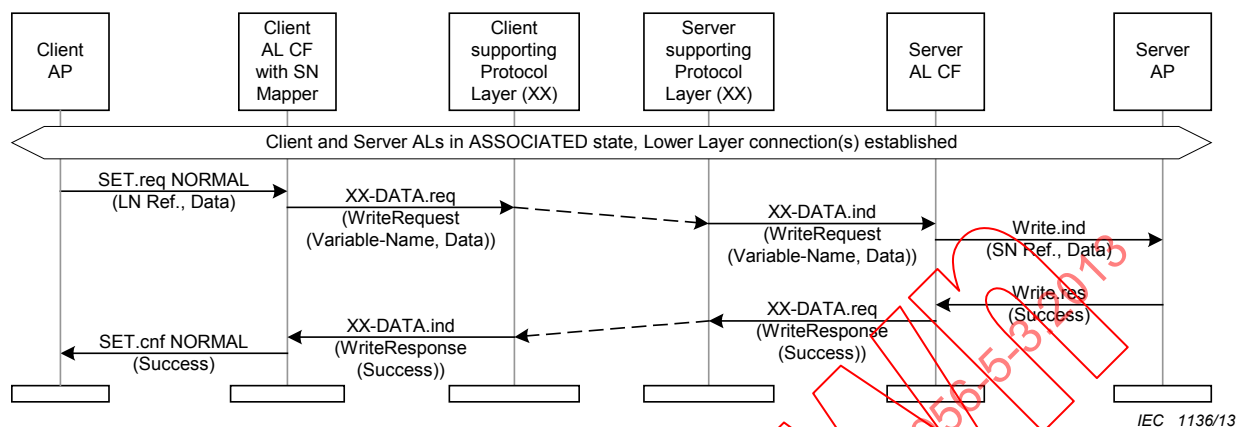


Figure 25 – MSC of the Write service used for writing an attribute

Figure 26 shows the MSC of a Write service used to invoke a single method, in the case of success.

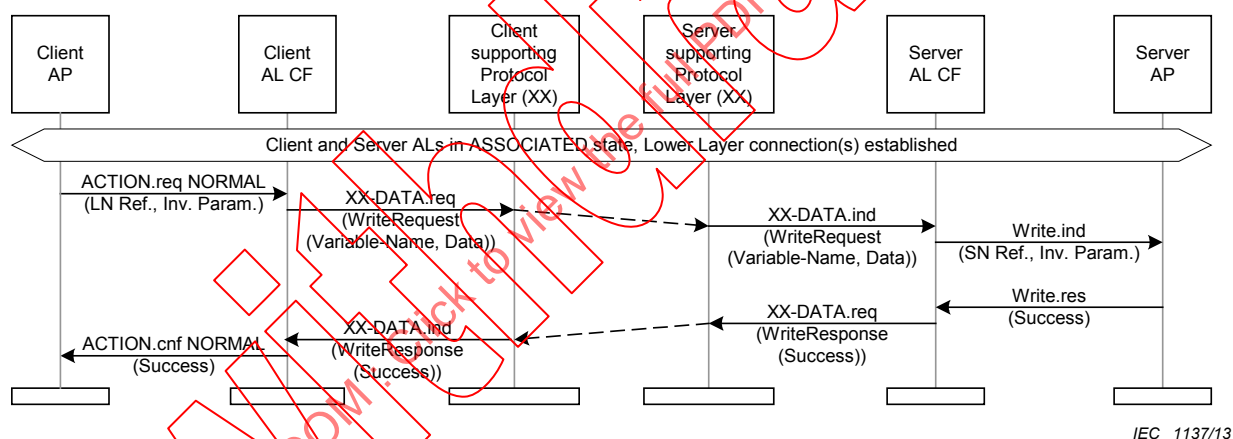


Figure 26 – MSC of the Write service used for invoking a method

Figure 27 shows the MSC of a Write service for writing a single attribute with the result returned in three blocks.

The process of preparing and transporting the long data is essentially the same as in the case of the SET and ACTION services:

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.

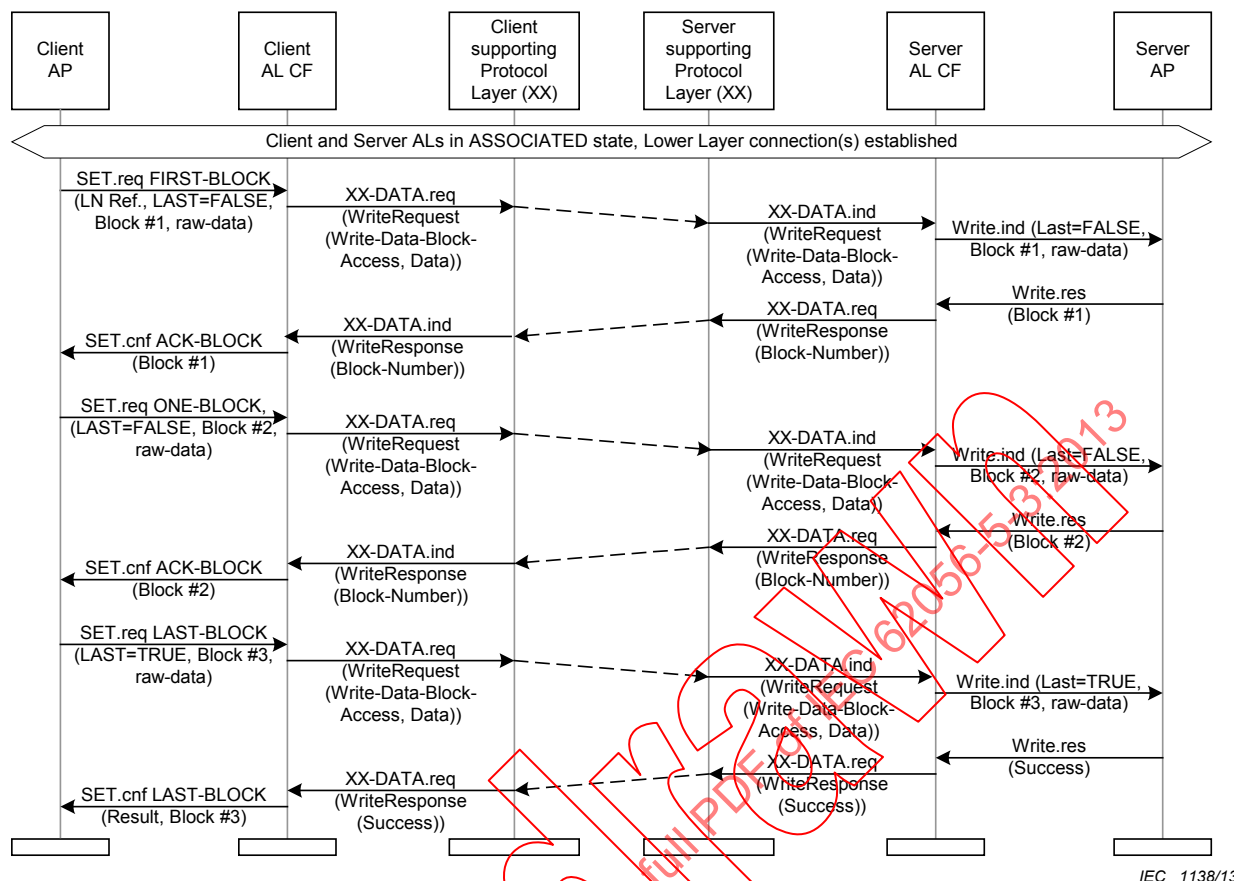


Figure 27 – MSC of the Write Service used for writing an attribute, with block transfer

7.3.9 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 (CO) or connectionless data (CL) services of the supporting protocol layer.

As explained in 6.14, 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 43;
- 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 44.

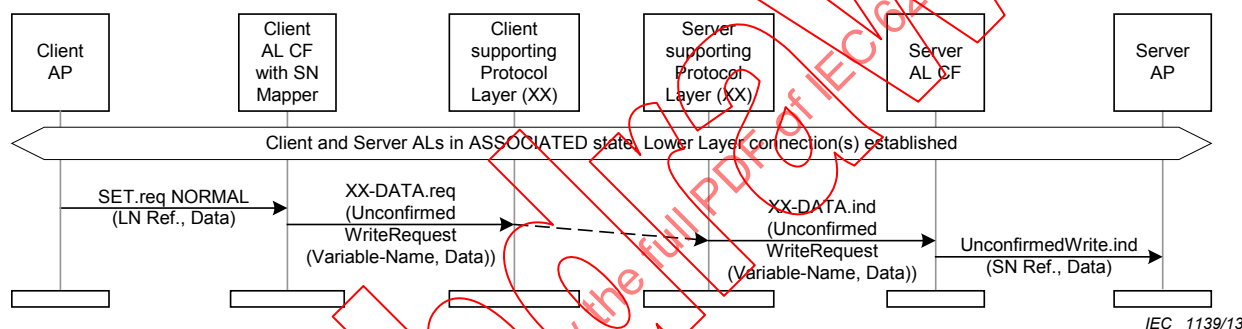
Table 43 – Mapping between the SET and the UnconfirmedWrite services

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 44 – Mapping between the ACTION and the UnconfirmedWrite services

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

**Figure 28 – MSC of the Unconfirmed Write service used for writing an attribute**

7.3.10 Protocol for the InformationReport service

The protocol for the InformationReport service, specified in 6.15, is essentially the same as that of the EventNotification service, see 7.3.6.

As the InformationReport service does not contain the optional Application_Addresses parameter, unlike the EventNotification service, 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 device, 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 Annex A.

The InformationReport service may carry several attribute names and their contents. On the other hand, the EventNotification service specified in 6.9 contains only one attribute reference. Therefore, when the InformationReportRequest APDU contains more than one attribute, it shall be mapped to several EventNotification.ind services, as shown in Table 45.

Table 45 – 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}

8 Abstract syntax of ACSE and COSEM APDUs

The abstract syntax of COSEM APDUs is specified in this Clause 8 using ASN.1. See ISO/IEC 8824-1.

COSEMpdu 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.
}
```

xDLMS-APDU ::= CHOICE

```
{
    -- standardised xDLMS pdus used in DLMS/COSEM
    -- standardized DLMS pdus

    -- 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]
    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,

-- the four ACSE APDUs
aarg                      AARQ-apdu,
aare                      AARE-apdu,
rlrq                      RLQR-apdu,          -- OPTIONAL
rlre                      RLRE-apdu          -- OPTIONAL

-- 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  EVENT-NOTIFICATION-Request,
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

-- reserved
-- reserved                [230] -- 0xE6
-- reserved                [231] -- 0xE7
}

```

AARQ-apdu ::= [APPLICATION 0] IMPLICIT SEQUENCE

```
{
-- [APPLICATION 0] == [ 60H ] = [ 96 ]

    protocol-version                [0] IMPLICIT BIT STRING {version1 (0)}
                                     DEFAULT {version1},
    application-context-name        [1] Application-context-name,
    called-AP-title                 [2] AP-title OPTIONAL,
    called-AE-qualifier             [3] AE-qualifier OPTIONAL,
    called-AP-invocation-id         [4] AP-invocation-identifier OPTIONAL,
    called-AE-invocation-id         [5] AE-invocation-identifier OPTIONAL,
    calling-AP-title               [6] AP-title OPTIONAL,
    calling-AE-qualifier            [7] AE-qualifier OPTIONAL,
    calling-AP-invocation-id        [8] AP-invocation-identifier OPTIONAL,
    calling-AE-invocation-id        [9] AE-invocation-identifier OPTIONAL,

-- The following field shall not be present if only the kernel is used.
    sender-acse-requirements        [10] IMPLICIT ACSE-requirements OPTIONAL,

-- The following field shall only be present if the authentication functional unit is
-- selected.
    mechanism-name                  [11] IMPLICIT Mechanism-name OPTIONAL,

-- The following field shall only be present if the authentication functional unit is
--selected.
    calling-authentication-value     [12] EXPLICIT Authentication-value OPTIONAL,
    implementation-information       [29] IMPLICIT Implementation-data OPTIONAL,
    user-information                [30] EXPLICIT Association-information OPTIONAL
}

-- The user-information field shall carry an xDLMS InitiateRequest APDU encoded in A-
XDR.
-- The resulting OCTET STRING shall be encoded 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. The resulting OCTET STRING shall be encoded 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,
    external                  [2] IMPLICIT    EXTERNAL,
    other                     [3] IMPLICIT    SEQUENCE
    {
        other-mechanism-name Mechanism-name,
        other-mechanism-value ANY DEFINED BY other-mechanism-name
    }
}

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),
        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)

-- IEC 62056-8-3 specifies the abstract syntax of the CIASE APDUs issued in the PLC S-
FSK -- profile.

-- 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 61334-6:2000, Annex C,
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),
    reserved-one           (1),
    reserved-two           (2),
    read                   (3),
    write                  (4),
    unconfirmed-write       (5),
    reserved-six           (6),
    reserved-seven         (7),
    attribute0-supported-with-set (8),
    priority-mgmt-supported (9),
    attribute0-supported-with-get (10),
    block-transfer-with-get-or-read (11),
    block-transfer-with-set-or-write (12),
    block-transfer-with-action (13),
    multiple-references     (14),
    information-report      (15),
    reserved-sixteen        (16),
    reserved-seventeen      (17),
    parameterized-access    (18),
    get                    (19),
    set                    (20),
    selective-access        (21),
    event-notification      (22),

```

```

        action                                     (23)
    }(SIZE(24))
    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,
        getTIAttribute                             [8] ServiceError,
        changeScope                                [9] ServiceError,
        start                                       [10] ServiceError,
        stop                                        [11] ServiceError,
        resume                                     [12] ServiceError,
        makeUsable                                 [13] ServiceError,
        initiateLoad                               [14] ServiceError,
        loadSegment                                [15] ServiceError,
        terminateLoad                              [16] ServiceError,
        initiateUpLoad                             [17] ServiceError,
        upLoadSegment                             [18] ServiceError,
        terminateUpLoad                           [19] ServiceError
    }

    ServiceError ::= CHOICE
    {
        application-reference [0] IMPLICIT ENUMERATED
        {
            -- DLMS provider only
            other (0),
            time-elapsed (1), -- time out since request sent
            application-unreachable (2), -- peer AEI not reachable
            application-reference-invalid (3), -- addressing trouble
            application-context-unsupported (4), -- application-context incompatibility
            provider-communication-error (5), -- error at the local or distant equipment
            deciphering-error (6) -- error detected by the deciphering function
        },

        hardware-resource [1] IMPLICIT ENUMERATED
        {
            -- VDE hardware troubles
            other (0),
            memory-unavailable (1),
            processor-resource-unavailable (2),
            mass-storage-unavailable (3),
            other-resource-unavailable (4)
        },

        vde-state-error [2] IMPLICIT ENUMERATED
        {
            -- Error source description
            other (0),
            no-dlms-context (1),
            loading-data-set (2),
            status-nochange (3),
            status-inoperable (4)
        },

        service [3] IMPLICIT ENUMERATED
        {
            -- service handling troubles
            other (0),
            pdu-size (1), -- pdu too long
            service-unsupported (2) -- as defined in the conformance block
        },
    }

```



```

definition
{
  -- object bound troubles in a service
  other
  object-undefined
  object-class-inconsistent
  object-attribute-inconsistent
},

access
{
  -- object access error
  other
  scope-of-access-violated
  object-access-violated
  hardware-fault
  object-unavailable
},

initiate
{
  -- initiate service error
  other
  dlms-version-too-low
  incompatible-conformance
  pdu-size-too-short
  refused-by-the-VDE-Handler
},

load-data-set
{
  -- data set load services error
  other
  primitive-out-of-sequence
  not-loadable
  dataset-size-too-large
  not-awaited-segment
  interpretation-failure
  storage-failure
  data-set-not-ready
},

-- change-scope
task
{
  -- TI services error
  other
  no-remote-control
  ti-stopped
  ti-running
  ti-unusable
},

-- other

```

[4] **IMPLICIT ENUMERATED**

(0),
 (1), -- object not defined at the VDE
 (2), -- class of object incompatible with asked
 -- service
 (3) -- object attributes are inconsistent

[5] **IMPLICIT ENUMERATED**

(0),
 (1), -- access denied through authorisation reason
 (2), -- access incompatible with object attribute
 (3), -- access fail for hardware reason
 (4) -- VDE hands object for unavailable

[6] **IMPLICIT ENUMERATED**

(0),
 (1), -- proposed DLMS version too low
 (2), -- proposed service not sufficient
 (3), -- proposed PDU size too short
 (4) -- vaa creation impossible or not allowed

[7] **IMPLICIT ENUMERATED**

(0),
 (1), -- according to the DataSet loading state
 -- transitions
 (2), -- loadable attribute set to FALSE
 (3), -- evaluated Data Set size too large
 (4), -- proposed segment not awaited
 (5), -- segment interpretation error
 (6), -- segment storage error
 (7) -- Data Set not in correct state for
 -- uploading

[8] **IMPLICIT ENUMERATED**

[9] **IMPLICIT ENUMERATED**

(0),
 (1), -- Remote Control parameter set to FALSE
 (2), -- TI in stopped state
 (3), -- TI in running state
 (4) -- TI in unusable state

[10] **IMPLICIT ENUMERATED**

-- COSEM APDUs using short name referencing

ReadRequest ::= SEQUENCE OF Variable-Access-Specification

ReadResponse ::= SEQUENCE OF CHOICE

```
{
  data [0] Data,
  data-access-error [1] IMPLICIT Data-Access-Result,
  data-block-result [2] IMPLICIT Data-Block-Result,
  block-number [3] IMPLICIT Unsigned16
}
```

WriteRequest ::= SEQUENCE

```
{
  variable-access-specification SEQUENCE OF Variable-Access-Specification,
  list-of-data SEQUENCE OF Data
}
```

WriteResponse ::= SEQUENCE OF CHOICE

```
{
  success [0] IMPLICIT NULL,
  data-access-error [1] IMPLICIT Data-Access-Result,
  block-number [2] Unsigned16
}
```

UnconfirmedWriteRequest ::= SEQUENCE

```
{
  variable-access-specification SEQUENCE OF Variable-Access-Specification,
  list-of-data SEQUENCE OF Data
}
```

InformationReportRequest ::= SEQUENCE

```
{
  current-time GeneralizedTime OPTIONAL,
  variable-access-specification SEQUENCE OF Variable-Access-Specification,
  list-of-data SEQUENCE OF Data
}
```

-- COSEM APDUs using logical name referencing

Get-Request ::= CHOICE

```
{
  get-request-normal [1] IMPLICIT Get-Request-Normal,
  get-request-next [2] IMPLICIT Get-Request-Next,
  get-request-with-list [3] IMPLICIT Get-Request-With-List
}
```

Get-Request-Normal ::= SEQUENCE

```
{
  invoke-id-and-priority Invoke-Id-And-Priority,
  cosem-attribute-descriptor Cosem-Attribute-Descriptor,
  access-selection Selective-Access-Descriptor OPTIONAL
}
```

Get-Request-Next ::= SEQUENCE

```
{
  invoke-id-and-priority Invoke-Id-And-Priority,
  block-number Unsigned32
}
```

Get-Request-With-List ::= SEQUENCE

```
{
  invoke-id-and-priority Invoke-Id-And-Priority,
  attribute-descriptor-list SEQUENCE OF Cosem-Attribute-Descriptor-With-Selection
}
```

Get-Response ::= CHOICE

```
{
  get-response-normal [1] IMPLICIT Get-Response-Normal,
  get-response-with-datablock [2] IMPLICIT Get-Response-With-Datablock,
  get-response-with-list [3] IMPLICIT Get-Response-With-List
}
```

Get-Response-Normal ::= SEQUENCE

```
{
  invoke-id-and-priority Invoke-Id-And-Priority,
  result Get-Data-Result
}
```

```

Get-Response-With-Datablock ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    result                      DataBlock-G
}

Get-Response-With-List        ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    result                      SEQUENCE OF Get-Data-Result
}

Set-Request                   ::= CHOICE
{
    set-request-normal          [1] IMPLICIT Set-Request-Normal,
    set-request-with-first-datablock [2] IMPLICIT Set-Request-With-First-Datablock,
    set-request-with-datablock   [3] IMPLICIT Set-Request-With-Datablock,
    set-request-with-list       [4] IMPLICIT Set-Request-With-List,
    set-request-with-list-and-first-datablock [5] IMPLICIT Set-Request-With-List-And-First-Datablock
}

Set-Request-Normal            ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    cosem-attribute-descriptor  Cosem-Attribute-Descriptor,
    access-selection            Selective-Access-Descriptor OPTIONAL,
    value                       Data
}

Set-Request-With-First-Datablock ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    cosem-attribute-descriptor  Cosem-Attribute-Descriptor,
    access-selection            Selective-Access-Descriptor OPTIONAL,
    datablock                  DataBlock-SA
}

Set-Request-With-Datablock    ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    datablock                  DataBlock-SA
}

Set-Request-With-List         ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    attribute-descriptor-list   SEQUENCE OF Cosem-Attribute-Descriptor-With-Selection,
    value-list                  SEQUENCE OF Data
}

Set-Request-With-List-And-First-Datablock ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    attribute-descriptor-list   SEQUENCE OF Cosem-Attribute-Descriptor-With-Selection,
    datablock                  DataBlock-SA
}

Set-Response                   ::= CHOICE
{
    set-response-normal          [1] IMPLICIT Set-Response-Normal,
    set-response-datablock       [2] IMPLICIT Set-Response-Datablock,
    set-response-last-datablock  [3] IMPLICIT Set-Response-Last-Datablock,
    set-response-last-datablock-with-list [4] IMPLICIT Set-Response-Last-Datablock-With-List,
    set-response-with-list       [5] IMPLICIT Set-Response-With-List
}

Set-Response-Normal           ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    result                      Data-Access-Result
}

Set-Response-Datablock        ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    block-number                Unsigned32
}

```

```

Set-Response-Last-Datablock ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    result                      Data-Access-Result,
    block-number                Unsigned32
}

Set-Response-Last-Datablock-With-List ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    result                      SEQUENCE OF Data-Access-Result,
    block-number                Unsigned32
}

Set-Response-With-List        ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    result                      SEQUENCE OF Data-Access-Result
}

Action-Request                ::= CHOICE
{
    action-request-normal      [1] IMPLICIT Action-Request-Normal,
    action-request-next-pblock [2] IMPLICIT Action-Request-Next-Pblock,
    action-request-with-list   [3] IMPLICIT Action-Request-With-List,
    action-request-with-first-pblock [4] IMPLICIT Action-Request-With-First-Pblock,
    action-request-with-list-and-first-pblock [5] IMPLICIT Action-Request-With-List-And-First-Pblock,
    action-request-with-pblock [6] IMPLICIT Action-Request-With-Pblock
}

Action-Request-Normal        ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    cosem-method-descriptor     Cosem-Method-Descriptor,
    method-invocation-parameters Data OPTIONAL
}

Action-Request-Next-Pblock   ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    block-number                Unsigned32
}

Action-Request-With-List     ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    cosem-method-descriptor-list SEQUENCE OF Cosem-Method-Descriptor,
    method-invocation-parameters SEQUENCE OF Data
}

Action-Request-With-First-Pblock ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    cosem-method-descriptor     Cosem-Method-Descriptor,
    pblock                     DataBlock-SA
}

Action-Request-With-List-And-First-Pblock ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    cosem-method-descriptor-list SEQUENCE OF Cosem-Method-Descriptor,
    pblock                     DataBlock-SA
}

Action-Request-With-Pblock   ::= SEQUENCE
{
    invoke-id-and-priority      Invoke-Id-And-Priority,
    pblock                     DataBlock-SA
}

Action-Response              ::= CHOICE
{
    action-response-normal      [1] IMPLICIT Action-Response-Normal,
    action-response-with-pblock [2] IMPLICIT Action-Response-With-Pblock,
    action-response-with-list   [3] IMPLICIT Action-Response-With-List,
    action-response-next-pblock [4] IMPLICIT Action-Response-Next-Pblock
}

```

```

Action-Response-Normal      ::= SEQUENCE
{
    invoke-id-and-priority    Invoke-Id-And-Priority,
    single-response           Action-Response-With-Optional-Data
}

Action-Response-With-Pblock ::= SEQUENCE
{
    invoke-id-and-priority    Invoke-Id-And-Priority,
    pblock                   DataBlock-SA
}

Action-Response-With-List   ::= SEQUENCE
{
    invoke-id-and-priority    Invoke-Id-And-Priority,
    list-of-responses         SEQUENCE OF Action-Response-With-Optional-Data
}

Action-Response-Next-Pblock ::= SEQUENCE
{
    invoke-id-and-priority    Invoke-Id-And-Priority,
    block-number              Unsigned32
}

EventNotificationRequest    ::= SEQUENCE
{
    time                      OCTET STRING OPTIONAL,
    cosem-attribute-descriptor Cosem-Attribute-Descriptor,
    attribute-value           Data
}

ExceptionResponse           ::= SEQUENCE
{
    state-error               [0] IMPLICIT ENUMERATED
    {
        service-not-allowed    (1),
        service-unknown        (2)
    },
    service-error             [1] IMPLICIT ENUMERATED
    {
        operation-not-possible (1),
        service-not-supported  (2),
        other-reason           (3)
    }
}

```

Variable-Access-Specification ::= **CHOICE**

Parameterized-Access ::= SEQUENCE

Block-Number-Access ::= **SEQUENCE**

Read-Data-Block-Access ::= **SEQUENCE**

```
Write-Data-Block-Access ::= SEQUENCE
```

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] is not used in DLMS/COSEM. It uses [23] and [24]
    octet-string [9] IMPLICIT OCTET STRING,
    visible-string [10] IMPLICIT VisibleString,
-- time [11] is not used in DLMS/COSEM. It uses [27]
    UTF8-string [12] IMPLICIT OCTET-STRING
-- An ordered sequence of characters encoded as UTF-8.
    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**

<pre> { null-data array { number-of-elements type-description }, structure boolean bit-string double-long double-long-unsigned octet-string visible-string UTF8-string bcd integer long unsigned long-unsigned long64 long64-unsigned enum float32 float64 date_time date time dont-care </pre>	<pre> [0] [1] [2] [3] [4] [5] [6] [9] [10] [12] [13] [15] [16] [17] [18] [20] [21] [22] [23] [24] [25] [26] [27] [255] </pre>	<pre> IMPLICIT NULL, IMPLICIT SEQUENCE Unsigned16, TypeDescription IMPLICIT SEQUENCE OF TypeDescription, IMPLICIT NULL </pre>

Data-Access-Result ::= **ENUMERATED**

<pre> { success hardware-fault temporary-failure read-write-denied object-undefined object-class-inconsistent object-unavailable type-unmatched scope-of-access-violated data-block-unavailable long-get-aborted no-long-get-in-progress long-set-aborted no-long-set-in-progress data-block-number-invalid other-reason </pre>	<pre> (0), (1), (2), (3), (4), (9), (11), (12), (13), (14), (15), (16), (17), (18), (19), (250) </pre>

Action-Result ::= **ENUMERATED**

<pre> { success hardware-fault temporary-failure read-write-denied object-undefined object-class-inconsistent object-unavailable type-unmatched scope-of-access-violated data-block-unavailable long-action-aborted no-long-action-in-progress other-reason </pre>	<pre> (0), (1), (2), (3), (4), (9), (11), (12), (13), (14), (15), (16), (250) </pre>

Invoke-Id-And-Priority ::= unsigned8

-- 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 IEC 62056-5-3, bits are numbered from 0 to 7.
 -- bit 7: priority, 0 = normal, 1 = high;
 -- bit 6: service-class, 0 = unconfirmed, 1 = confirmed;
 -- bits 5 to 4: reserved, shall be set to 0;
 -- bits 3 to 0: invoke-id, 0x0..0xF


```

-- Examples for the Invoke-Id-And-Priority byte:
-- Priority = High, Service-class = unconfirmed, Invoke-id = 1: -- value = 0x81
-- Priority = High, Service-class = confirmed, Invoke-id = 1: -- value = 0xC1
-- Priority = Normal, Service-class = confirmed, Invoke-id = 1: -- value = 0x41

Cosem-Attribute-Descriptor ::= SEQUENCE
{
    class-id                Cosem-Class-Id,
    instance-id             Cosem-Object-Instance-Id,
    attribute-id            Cosem-Object-Attribute-Id
}

Cosem-Method-Descriptor   ::= SEQUENCE
{
    class-id                Cosem-Class-Id,
    instance-id             Cosem-Object-Instance-Id,
    method-id              Cosem-Object-Method-Id
}

Cosem-Class-Id            ::= Unsigned16

Cosem-Object-Instance-Id  ::= OCTET STRING(SIZE(6))

Cosem-Object-Attribute-Id ::= Integer8

Cosem-Object-Method-Id    ::= Integer8

Selective-Access-Descriptor ::= SEQUENCE
{
    access-selector        Unsigned8,
    access-parameters      Data
}

Cosem-Attribute-Descriptor-With-Selection ::= SEQUENCE
{
    cosem-attribute-descriptor Cosem-Attribute-Descriptor,
    access-selection           Selective-Access-Descriptor OPTIONAL
}

Get-Data-Result           ::= CHOICE
{
    data                   [0] Data,
    data-access-result     [1] IMPLICIT Data-Access-Result
}

Data-Block-Result        ::= SEQUENCE -- Used in ReadResponse with block transfer
{
    last-block             BOOLEAN,
    block-number           Unsigned16,
    raw-data               OCTET STRING
}

DataBlock-G              ::= SEQUENCE -- G == DataBlock for the GET-response
{
    last-block             BOOLEAN,
    block-number           Unsigned32,
    result                 CHOICE
    {
        raw-data           [0] IMPLICIT OCTET STRING,
        data-access-result [1] IMPLICIT Data-Access-Result
    }
}

DataBlock-SA ::= SEQUENCE -- SA == DataBlock for the SET-request, ACTION-request and
-- ACTION-response
{
    last-block             BOOLEAN,
    block-number           Unsigned32,
    raw-data               OCTET STRING
}

Action-Response-With-Optional-Data ::= SEQUENCE
{
    result                 Action-Result,
    return-parameters      Get-Data-Result OPTIONAL
}

END

```

Annex A

(normative)

Using the COSEM application layer in various communications profiles

A.1 General

As it will be explained in IEC 62056-1-0¹⁸, the COSEM interface model for energy metering equipment, specified in IEC 62056-6-2 has been designed for use with a variety of communication profiles for exchanging data over various communication media. As it will be shown in IEC 62056-1-0, in each such profile, the application layer is the COSEM AL, providing the xDLMS services to access attributes and methods of COSEM objects. For each communication profile, the following elements shall be specified:

- the targeted communication environments;
- the structure of the profile (the set of protocol layers);
- the identification / addressing scheme;
- mapping of the COSEM AL services to the service set provided and used by the supporting layer;
- communication profile specific parameters of the COSEM AL services;
- other specific considerations / constraints for using certain services within a given profile.

A.2 Targeted communication environments

This part identifies the communication environments, for which the given communication profile is specified.

A.3 The structure of the profile

This part specifies the protocol layers included in the given profile.

A.4 Identification and addressing schemes

This part describes the identification and addressing schemes specific for the profile.

As described in IEC 62056-6-2:—, 4.7, metering equipment are modelled in COSEM as physical devices, containing one or more logical devices. In the COSEM client/server type model, data exchange takes place within application associations, 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 to exchange data with the help of the supporting lower layer protocols, the client- and server APs shall 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;

¹⁸ Under consideration.

The client- and server APs also identify the AAs.

A.5 Supporting layer services and service mapping

This part specifies the service mapping between the services requested by the COSEM AL and the services provided by its supporting layer.

In each communication profile, the 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, MSCs are generally used, showing the sequence of the events following a service invocation by the COSEM AP.

A.6 Communication profile specific parameters of the COSEM AL services

In COSEM, only the COSEM-OPEN service has communication profile specific parameters (the Protocol_Connection_Parameters). Their values and use are defined as part of the communication profile specification.

A.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.

Annex B (informative)

AARQ and AARE encoding examples

B.1 General

This annex contains examples of encoding of the AARQ and AARE APDUs, in cases of using various levels of authentication and in cases of success and failure.

In DLMS/COSEM, the AARQ, AARE, RLRQ and RLRE APDUs – see 7.2 – shall be encoded in BER (ISO/IEC 8825-1). 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.

B.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 B.1.

Table B.1 – Conformance block

Conformance::= [APPLICATION 31] IMPLICIT BIT STRING (SIZE(24))		LN referencing		SN referencing	
-- the bit is set when the corresponding service or functionality is available	Used with	Proposed/ Negotiated		Proposed / Negotiated	
reserved-zero (0),		0	0	0	0
reserved-one (1),		0	0	0	0
reserved-two (2),		0	0	0	0
read (3),	SN	0	0	1	1
write (4),	SN	0	0	1	1
unconfirmed-write (5),	SN	0	0	1	1
reserved-six (6),		0	0	0	0
reserved-seven (7),		0	0	0	0
attribute0-supported-with-set (8),	LN	0	0	0	0
priority-mgmt-supported (9),	LN	1	1	0	0
attribute0-supported-with-get (10),	LN	1	0	0	0
block-transfer-with-get-or-read (11),	LN	1	1	0	0
block-transfer-with-set-or-write (12),	LN	1	0	0	0
block-transfer-with-action (13),	LN	1	0	0	0
multiple-references (14),	LN/SN	1	0	1	1
information-report (15),	SN	0	0	1	1
reserved-sixteen (16),		0	0	0	0
reserved-seventeen (17),		0	0	0	0
parameterized-access (18),	SN	0	0	1	1
get (19),	LN	1	1	0	0
set (20),	LN	1	1	0	0
selective-access (21),	LN	1	1	0	0
event-notification (22),	LN	1	1	0	0
action (23),	LN	1	1	0	0
Value of the bit string		00 7E 1F	00 50 1F	1C 03 20	1C 03 20

With these parameters, the A-XDR encoding of the xDLMS InitiateRequest APDU is the following (see Table B.2):

Table B.2 – A-XDR encoding of the xDLMS InitiateRequest APDU

-- A-XDR encoding of the xDLMS InitiateRequest APDU	LN referencing	SN referencing
// encoding of the tag of the DLMS 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 61334-6:2000, Annex C, Examples 1 and 2, the proposed-conformance element of the xDLMS InitiateRequest APDU and the negotiated-conformance element of the xDLMS InitiateResponse APDU are encoded in BER. That's why the length of the bit-string and the number of the unused bits are encoded. ² For encoding of identifier octets see ISO/IEC 8825-1:2008, 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 the following (see Table B.3):

Table B.3 – A-XDR encoding of the xDLMS InitiateResponse APDU

-- A-XDR encoding of the xDLMS InitiateResponse APDU	LN referencing	SN referencing
// encoding of the tag of the DLMS 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

B.3 Specification of the AARQ and AARE APDUs

AARQ-apdu ::= [APPLICATION 0] **IMPLICIT SEQUENCE**

```
{
-- [APPLICATION 0] == [ 60H ] = [ 96 ]

    protocol-version                [0] IMPLICIT BIT STRING {version1 (0)}
                                     DEFAULT {version1},
    application-context-name        [1] Application-context-name,
    called-AP-title                 [2] AP-title OPTIONAL,
    called-AE-qualifier             [3] AE-qualifier OPTIONAL,
    called-AP-invocation-id         [4] AP-invocation-identifier OPTIONAL,
    called-AE-invocation-id         [5] AE-invocation-identifier OPTIONAL,
    calling-AP-title               [6] AP-title OPTIONAL,
    calling-AE-qualifier           [7] AE-qualifier OPTIONAL,
    calling-AP-invocation-id       [8] AP-invocation-identifier OPTIONAL,
    calling-AE-invocation-id       [9] AE-invocation-identifier OPTIONAL,

-- The following field shall not be present if only the kernel is used.
    sender-acse-requirements       [10] IMPLICIT ACSE-requirements OPTIONAL,

-- The following field shall only be present if the authentication functional unit is
-- selected.
    mechanism-name                 [11] IMPLICIT Mechanism-name OPTIONAL,
```



```
-- The following field shall only be present if the authentication functional unit is
--selected.
    calling-authentication-value      [12] EXPLICIT    Authentication-value OPTIONAL,
    implementation-information        [29] IMPLICIT    Implementation-data OPTIONAL,
    user-information                  [30] EXPLICIT    Association-information OPTIONAL
}

-- The user-information field shall carry an xDLMS InitiateRequest APDU encoded in A-
XDR.
-- The resulting OCTET STRING shall be encoded 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. The resulting OCTET STRING shall be encoded in BER.
```

B.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 fields of the AARE are not present;
- the value of the mechanism-name:
 - in the case of low-level-security: 2, 16, 756, 5, 8, 2, 1;
 - in the case of high-level-security (5): 2, 16, 756, 5, 8, 2, 5;
- the calling-authentication-value:
 - in the case of low-level-security is 12345678 (encoded as 31 32 33 34 35 36 37 38);
 - in the case of high-level security, (challenge CtoS) is K56iVagY (encoded as 4B 35 36 69 56 61 67 59);
- the responding authentication-value (challenge StoC) is P6wRJ21F (encoded as 50 36 77 52 4A 32 31 46);

- the optional implementation-information field in the AARQ and AARE APDUs is not present;
- the user-information field carries the xDLMS InitiateRequest / InitiateResponse APDUs as shown above.

The application-context-name and the (authentication) mechanism-name OBJECT IDENTIFIERS are encoded as follows:

- BER Encoding for OBJECT IDENTIFIER is a packed sequence of numbers representing the arc labels. Each number – except the first two, which are combined into one – is represented as a series of octets, with 7 bits being used from each octet and the most significant bit is set to 1 in all but the last octet. The fewest possible number of octets shall 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 $40 \times \text{First} + \text{Second} \rightarrow 40 \times 2 + 16 = 96 = 0x60$;
 - 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.

B.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 B.4. See also Table B.5.

Table B.4 – BER encoding of the AARQ APDU

-- BER encoding of 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 of 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 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	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)						
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

-- BER encoding of the AARQ APDU		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 :									
- low-level-security-mechanism-name (1),		-	60 85 74 05 08 02 01	60 85 74 05 08 02 05		-	60 85 74 05 08 02 01	60 85 74 05 08 02 05	
- high-level-security-mechanism-name (5)									
-- calling-authentication-value field ([12], Authentication-value 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 (charsstring [0])		-	80	80		-	80	80	
IMPLICIT GraphicString									
// encoding of the length of the Authentication-value's value field (8 octets)			08	08		-	08	08	
// encoding of the calling-authentication-value:									
- in the case of LLS, the value of the Password "12345678"		-	31 32 33 34 35 36 37 38	4B 35 36 69 56 61 67 59		-	31 32 33 34 35 36 37 38	4B 35 36 69 56 61 67 59	
- in the case of HLS, the value of challenge Ctos "K56iVagY"									
-- encoding of the user-information field component (Association-information, OCTET STRING)									
// encoding of 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 of 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	06 5F 1F 04 00 00 7E 1F 04 B0	06 5F 1F 04 00 00 7E 1F 04 B0		01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0	06 5F 1F 04 00 00 7E 1F 04 B0	06 5F 1F 04 00 00 7E 1F 04 B0	

Table B.5 – 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

B.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 B.6. See also Table B.7.

Table B.6 – BER encoding of the AARE APDU

-- BER encoding of 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], Application)	61				61	
// encoding of the length of the AARE's contents 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 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: NOTE when the proposed application-context does not fit the application-context supported by the server, the server responds either 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 of 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;							
- failure 1: 2, application-context-name not supported;		00	02	01	0E	00	0E
- failure 2: 1, no-reason-given.							
- success, HLS security (5): 14, authentication required;							
-- encoding of 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	-				07	-	07

-- BER encoding of 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		
byte of the BIT STRING								
// 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 of 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 of the user-information field component (Association-information, OCTET STRING)								
// encoding of the tag for the user-information field component ([30], Context-specific, Constructed)	BE	BE	BE	BE	BE	BE		
// encoding of the length of the tagged component's value field	10	10	06	10	10	10		
// encoding of the choice for user-information (OCTET STRING, Universal)	04	04	04	04	04	04		
// encoding of the length of the OCTET STRING's value field	0E	0E	04	0E	0E	0E		

-- BER encoding of 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
// 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 B.7 – 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

Annex C (informative)

Encoding examples: AARQ and AARE APDUs using a ciphered application context

C.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 are 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 1200 bytes (0x04B0).

The A-XDR encoding of the xDLMS InitiateRequest APDU carrying a dedicated key is shown in Table C.1.

Table C.1 – A-XDR encoding of the xDLMS InitiateRequest APDU

// encoding of the tag of the DLMS APDU CHOICE (InitiateRequest)	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 61334-6:2000, Annex C, Examples 1 and 2, the proposed-conformance element of the xDLMS InitiateRequest APDU and the negotiated-conformance element of the xDLMS InitiateResponse APDU are encoded in BER. That's why the length of the bit-string and the number of the unused bits are encoded.
- ² For encoding of identifier octets see ISO/IEC 8825-1:2008, 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.

C.2 Authenticated encryption of the xDLMS InitiateRequest APDU

Table C.2 shows the encoding of an xDLMS InitiateRequest APDU which is also authenticated and encrypted.

Table C.2 – 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>	4D4D4D 0000BC614E (here, the five last octets contain the manufacturing number in hexadecimal form)	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>	000102030405060708090A0B0C0D0E0F	16	128
Authentication Key	<i>AK</i>	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	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 = SC-AE</i> <i>FC</i> 3001234567	5	40
Inputs				
xDLMS APDU to be protected	<i>ARDU</i>	01011000112233445566778899AABBCC DDEEFF0000065F1F0400007E1F04B0	31	188
Plaintext	<i>P</i>	01011000112233445566778899AABBCC DDEEFF0000065F1F0400007E1F04B0	31	188
Associated data	<i>A</i>	<i>SC</i> <i>AK</i> 30D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	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

C.3 The AARQ APDU

In this example, the following values are used:

- Application-Context-Name: Logical_Name_Referencing_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 C.3.

Table C.3 – 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) 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 of the calling-authentication-value (12345678)	3132333435363738
-- encoding of 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

C.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 1 024 bytes (0x0400).

The A-XDR encoding of the xDLMS InitiateResponse APDU is shown in Table C.4.

Table C.4 – A-XDR encoding of the xDLMS InitiateResponse APDU

// encoding of the tag of the DLMS 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	0800065F1F040000 7C1F04000007

C.5 Authenticated encryption of the xDLMS InitiateResponse APDU

Table C.5 shows the encoding of the xDLMS InitiateResponse APDU which is also authenticated and encrypted.

Table C.5 – 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 hexadecimal form)	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>	000102030405060708090A0B0C0D0E0F	16	128
Authentication Key	<i>AK</i>	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	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 = 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> 30D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF	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

C.6 The AARE APDU

The BER encoding of the AARE APDU is shown in Table C.6.

Table C.6 – 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 contents 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 responds either 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 of 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 of 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 23

C.7 The RLRQ APDU (carrying a ciphered xDLMS InitiateRequest APDU)

The BER encoding of the RLRQ APDU is shown in Table C.7.

Table C.7 – 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 of 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

C.8 The RLRE APDU (carrying a ciphered xDLMS InitiateResponse APDU)

The BER encoding of the RLRQ APDU is shown in Table C.8.

Table C.8 – 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 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 of 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

Annex D (informative)

Data transfer service examples

Tables D.2 to D.9 show examples for data exchange using xDLMS services with LN referencing (left column) and SN referencing (right column). Table D.1 shows the objects used in the examples.

Table D.1 – 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.

NOTE What is negotiated is the APDU size, not the block size. Therefore, the block size is smaller than the APDU size.

Table D.2 – Example: Reading the value of a single attribute without block transfer

<pre>C001C1 00010000800000FF0200 <GetRequest> <GetRequestNormal> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0000800000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </GetRequestNormal> </GetRequest></pre>	<pre>0501 020100 <ReadRequest Qty="0001" > <VariableName Value="0100" /> </ReadRequest></pre>
<pre>C401C1 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 <GetResponse> <GetResponseNormal> <InvokeIdAndPriority Value="C1" /> <Result> <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> </Result> </GetResponseNormal> </GetResponse></pre>	<pre>0C01 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 <ReadResponse Qty="0001" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> </ReadResponse></pre>

Table D.3 – Example: Reading the value of a list of attributes without 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> </GetRequest> </pre>	<pre> 0502 020100 020110 <ReadRequest Qty="0002" > <VariableName Value="0100" /> <VariableName Value="0110" /> </ReadRequest> </pre>
<pre> C403C1 02 00 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 00 0A03 303030 <GetResponse> <GetResponseWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> </Result> </GetResponseWithList> </GetResponse> </pre>	<pre> 0C02 00 00 0932 01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950 00 0A03 303030 <ReadResponse Qty="0002" > <Data> <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> </Data> </ReadResponse> </pre>

<pre></Data> <Data> <VisibleString Value="303030" /> </Data> </Result> </GetResponseWithList> </GetResponse></pre>	<pre>33343536373839404142434445464748 4950" /> <VisibleString Value="303030" /> </Data> </ReadResponse></pre>

Table D.4 – Example. Reading the value of a single attribute with block transfer

<pre>C001C1 00010000800000FF0200 <GetRequest> <GetRequestNormal> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0008000000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </GetRequestNormal> </GetRequest></pre>	<pre>0501 020100 <ReadRequest Qty="0001" > <VariableName Value="0100" /> </ReadRequest></pre>
<pre>C402C1 00 00 00000001 00 00 1E 093201020304050607080910111213 141516171819202122232425262728 <GetResponse> <GetResponseWithDataBlock> <InvokeIdAndPriority Value="C1" /> <Result> <LastBlock Value="00" /> <BlockNumber Value="00000001" /> <Result> <RawData Value="09320102030405060708091011121314 1516171819202122232425262728" /> </Result> </Result> </GetResponseWithDataBlock> </GetResponse> // 30 bytes of raw-data contains data type, length and 28 bytes</pre>	<pre>0001 0000 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</pre>

<pre>// of data. Note that Data is encoded, not Get-Data-result. C002C1 00000001 <GetRequest> <GetRequestForNextDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </GetRequestForNextDataBlock> </GetRequest></pre>	<pre>// block transfer, the number of results is encoded because the // ReadResponse is a SEQUENCE OF CHOICE. 0501 050001 <ReadRequest Qty="0001" > <BlockNumberAccess> <BlockNumber Value="0001" /> </BlockNumberAccess> </ReadRequest></pre>
<pre>C402C1 01 02 00000002 00 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>00C01 02 01 0002 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 D.5 – 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="0008000000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </_AttributeDescriptorWithSelection> <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0008001000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> </_AttributeDescriptorWithSelection> </AttributeDescriptorList> </GetRequest> </pre>	<pre> 0502 020100 020110 <ReadRequest Qty="0002" > <VariableName Value="0100" /> <VariableName Value="0110" /> </ReadRequest> </pre>
<pre> C402C1 00 00000001 00 00 1E 02000932010203040506070809101112 1314151617181920212223242526 <GetResponse> <GetResponseWithDataBlock> <InvokeIdAndPriority Value="C1" /> <Result> <LastBlock Value="00" /> <BlockNumber Value="00000001" /> <Result> <RawData Value="02000932010203040506070809101112 1314151617181920212223242526" /> </Result> </Result> </GetResponse> </pre>	<pre> 0C01 02 00 0001 21 02000932010203040506070809101112 13141516171819202122232425262728 29 <ReadResponse Qty="0001" > <DataBlockResult> <LastBlock Value="00" /> <BlockNumber Value="00000001" /> <RawData Value="02000932010203040506070809101112 13141516171819202122232425262728 29" /> </DataBlockResult> </ReadResponse> </pre>

<pre> </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> // 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> <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> 0201 02 01 0002 1E 30313233343536373839404142434445 4647484950000A03303030 <ReadResponse Qty="0001" > <DataBlockResult> <LastBlock Value="01" /> <BlockNumber Value="0002" /> <RawData Value="30313233343536373839404142434445 4647484950000A03303030" /> </DataBlockResult> </ReadResponse> // The APDU is 34 bytes. It contains the second and last block. // 27 bytes of raw-data contains the remaining 21 bytes of the // first result and the six bytes of the second result: success, // visible string of 3 elements </pre>

Table D.6 – 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>00D01 00 <WriteResponse Qty="0001" > <Success /> </WriteResponse></pre>

Table D.7 – Example: Writing the value of a list of attributes without block transfer

C104C1	0602
02	020100
00010000800000FF0200	020110
00010000800100FF0200	02
02	0932
0932	01020304050607080910111213141516
01020304050607080910111213141516	17181920212223242526272829303132
17181920212223242526272829303132	33343536373839404142434445464748
33343536373839404142434445464748	4950
4950	0A03
0A03	303030
303030	
<SetRequest>	<WriteRequest>
<SetRequestNormalWithList>	<ListOfVariableAccessSpecification Qty="0002" >
<InvokeIdAndPriority Value="C1" />	<VariableName Value="0100" />
<AttributeDescriptorList Qty="0002" >	<VariableName Value="0110" />
<_AttributeDescriptorWithSelection>	</ListOfVariableAccessSpecification>
<AttributeDescriptor>	<ListOfData Qty="0002" >
<ClassId Value="0001" />	<OctetString Value="01020304050607080910111213141516
<InstanceId Value="000080000000FF" />	17181920212223242526272829303132
<AttributeId Value="02" />	33343536373839404142434445464748
</AttributeDescriptor>	4950" />
</_AttributeDescriptorWithSelection>	<VisibleString Value="303030" />
<_AttributeDescriptorWithSelection>	</ListOfData>
<AttributeDescriptor>	</WriteRequest>
<ClassId Value="0001" />	
<InstanceId Value="0000800100FF" />	
<AttributeId Value="02" />	
</AttributeDescriptor>	
</_AttributeDescriptorWithSelection>	

<pre></AttributeDescriptorList> <ValueList Qty="0002" > <OctetString Value="01020304050607080910111213141516 17181920212223242526272829303132 33343536373839404142434445464748 4950" /> <VisibleString Value="303030" /> </ValueList> </SetRequestNormalWithList> </SetRequest></pre>	
<pre>C505C1 02 00 00 00 <SetResponse> <SetResponseWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <_DataAccessResult Value="Success" /> <_DataAccessResult Value="Success" /> </Result> </SetResponseWithList> </SetResponse></pre>	<pre>0002 00 00 00 <WriterResponse Qty="0002" > <Success /> <Success /> </WriterResponse></pre>

Table D.8 – Example: Writing the value of a single attribute with block transfer

C102C1 00010000800000FF0200 00 00000001 15 09320102030405060708091011121314 1516171819 <SetRequest> <SetRequestWithFirstDataBlock> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="000080000000FF" /> <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.	0601 07 00 0001 01 091F 010201000010932010203040506070809 101112131415161718192021222324 <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="00" /> <BlockNumber Value="0001" /> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> <ListOfSize Qty="0001" > <OctetString Value="010201000010932010203040506070809 101112131415161718192021222324" /> </ListOfSize> </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.
C502C1 00000001 <SetResponse> <SetResponseForDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000001" /> </SetResponseForDataBlock> </SetResponse>	0D01 02 0001 <WriteResponse Qty="0001" > <BlockNumber Value="0001" /> </WriteResponse>

<pre> C103C1 01 00000002 1F 2021223242526272829303132333435 363738394041424344454647484950 <SetRequest> <SetRequestWithDataBlock> <InvokeIdAndPriority Value="C1" /> <DataBlock> <LastBlock Value="01" /> <BlockNumber Value="00000002" /> <RawData Value="2021223242526272829303132333435 363738394041424344454647484950" /> </DataBlock> </SetRequestWithDataBlock> </SetRequest> // 31 bytes of raw-data contains the remaining 21 bytes of the // data to be written. </pre>	<pre> 0601 07 01 0002 01 091A 25262728293031323334353637383940 41424344454647484950 <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="01" /> <BlockNumber Value="0002" /> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> </ListOfData Qty="0001" > <OctetString Value="25262728293031323334353637383940 41424344454647484950" /> </ListOfData> </WriteRequest> // The APDU is 35 bytes. 26 bytes of octet-string contains raw- // data: the remaining 26 bytes of data to be written. 000100 </pre>
<pre> C503C10000000002 <SetResponse> <SetResponseForLastDataBlock> <InvokeIdAndPriority Value="C1" /> <Result Value="Success" /> <BlockNumber Value="00000002" /> </SetResponseForLastDataBlock> </SetResponse> </pre>	<pre> <WriteResponse Qty="0001" > <Success /> </WriteResponse> </pre>

Table D.9 – Example: Writing the value of a list of attributes with block transfer

<pre> C105C1 02 00010000800000FF0200 00010000800100FF0200 00 00 00000001 0A 02093201020304050607 <SetRequest> <SetRequestWithListAndWithFirstDataBlock> <InvokeIdAndPriority Value="C1" /> <AttributeDescriptorList Qty="0002" > <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="0008000000FF" /> <AttributeId Value="02" /> </AttributeDescriptor> <_AttributeDescriptorWithSelection> <AttributeDescriptor> <ClassId Value="0001" /> <InstanceId Value="000800100FF" /> <AttributeId Value="02" /> </AttributeDescriptor> <_AttributeDescriptorWithSelection> <AttributeDescriptor> <_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> 0601 07 00 0001 01 091F 02020100020110020932010203040506 070809101112131415161718192021 <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="00" /> <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 242526272829303132333435363738" /> </DataBlock> </SetRequestWithDataBlock> </SetRequest> // The APDU is 40 bytes. 31 bytes of raw-data contain the second // part of data to be written.	0601 07 00 00000002 01 091F 22232425262728293031323334353637 383940414243444546474849500A03 <WriteRequest> <ListOfVariableAccessSpecification Qty="0001" > <WriteDataBlockAccess> <LastBlock Value="00" /> <BlockNumber Value="0002" /> </WriteDataBlockAccess> </ListOfVariableAccessSpecification> </ListOfData> <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.
C502C100000002 <SetResponse> <SetResponseForDataBlock> <InvokeIdAndPriority Value="C1" /> <BlockNumber Value="00000002" /> </SetResponseForDataBlock> </SetResponse>	0D01 02 0002 <WriteResponse Qty="0001" > <BlockNumber Value="0002" /> </WriteResponse>

<pre> C103C1 01 00000003 11 3940414243444546474849500A033030 30 <SetRequest> <SetRequestWithDataBlock> <InvokeIdAndPriority Value="C1" /> <DataBlock> <LastBlock Value="01" /> <BlockNumber Value="00000003" /> <RawData Value="3940414243444546474849500A033030 30" /> </DataBlock> </SetRequestWithDataBlock> </SetRequest> // The APDU is 26 bytes. 17 bytes of raw-data contain the third // part of the first data and the second data to be written. </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" > <OctetStringValue="303030" /> </ListOfData> </WriteRequest> // The APDU is 12 bytes. 3 bytes of octet-string contains // raw-data: the value of the second attribute. </pre>
<pre> C504C1 02 00 00 00 00000003 <SetResponse> <SetResponseForLastDataBlockWithList> <InvokeIdAndPriority Value="C1" /> <Result Qty="0002" > <DataAccessResult Value="Success" /> <_DataAccessResult Value="Success" /> </Result> <BlockNumber Value="00000003" /> </SetResponseForLastDataBlockWithList> </SetResponse> </pre>	<pre> 0502 00 00 00 <WriteResponse Qty="0002" > <Success /> <Success /> </WriteResponse> </pre>

Annex E (informative)

Overview of cryptography

NOTE Annex E is provided as a brief introduction to the cryptography tools selected. More information can be found in the documents referenced.

E.1 General

NOTE The following text is quoted from NIST SP 800-21:2005, 3.1.

Cryptography is a branch of mathematics that is based on the transformation of data and can be used to provide several security services: confidentiality, data integrity, authentication, authorization and non-repudiation. Cryptography relies upon two basic components: an *algorithm* (or cryptographic methodology) and a *key*. The algorithm is a mathematical function, and the key is a parameter used in the transformation.

A cryptographic algorithm and a 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, symmetric key algorithms and asymmetric key algorithms:

- cryptographic hash functions 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 E.2;
- symmetric algorithms (often called secret key algorithms) use a single key to both apply the protection and to remove or check the protection. See E.3;
- asymmetric algorithms (often called public key algorithms) use two keys (i.e., a key pair): a public key and a private key that are mathematically related to each other. See E.4.

In order to use cryptography, cryptographic keys shall be “in place”, i.e., keys shall be established for parties using cryptography.

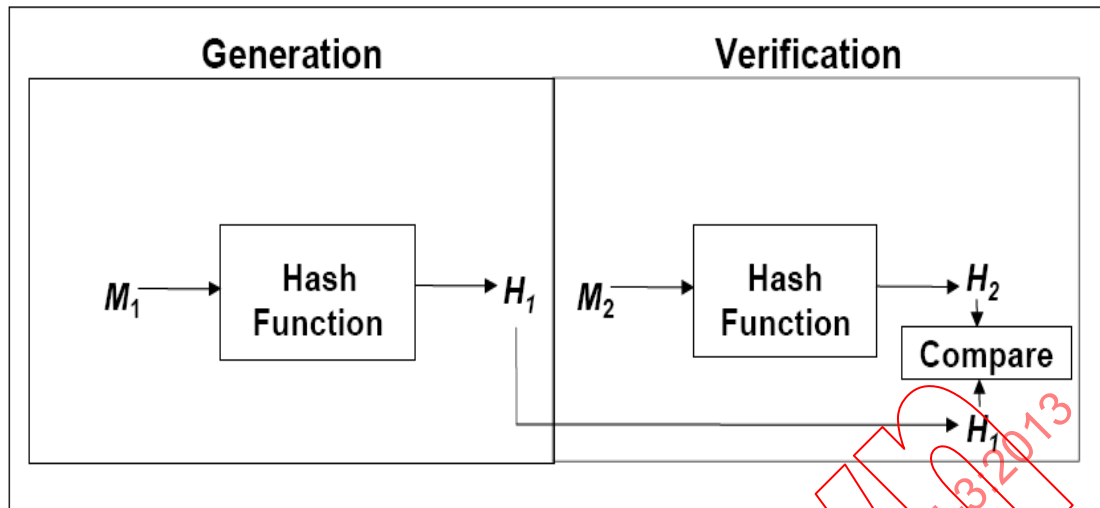
E.2 Hash functions

NOTE The following text is quoted from NIST SP 800-21:2005, 3.2.

A hash function produces a short representation of a longer message. A good hash function is a one-way function: it is easy to compute the hash value from a particular input; however, backing up the process from the hash value back to the input is extremely difficult. With a good hash function, it is also extremely difficult to find two specific inputs that produce the same hash value. Because of these characteristics, hash functions are often used to determine whether or not data has changed.

A hash function takes an input of arbitrary length and outputs a fixed length value. Common names for the output of a hash function include *hash value* and *message digest*. Figure E.1 depicts the use of a hash function.

A hash value (H1) is computed on data (M1). M1 and H1 are then saved or transmitted. At a later time, the correctness of the retrieved or received data is checked by labelling the received data as M2 (rather than M1) and computing a new hash value (H2) on the received value. If the newly computed hash value (H2) is equal to the retrieved or received hash value (H1), then it can be assumed that the retrieved or received data (M2) is the same as the original data (M1) (i.e. $M1 = M2$).



IEC 1140/13

Figure E.1 – Hash function

E.3 Symmetric key algorithms

E.3.1 General

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.

Symmetric key algorithms (often called secret key algorithms) use a single key to both apply the protection and to remove or check the protection. For example, the key used to encrypt data is also used to decrypt the encrypted data. This key shall be kept secret if the data is to retain its cryptographic protection. Symmetric 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).

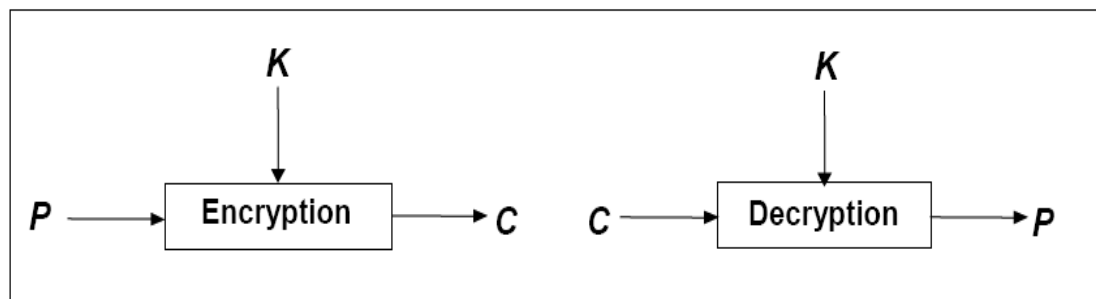
E.3.2 Encryption and decryption

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.1. In this context, “for Federal Government use” means “for the purposes of this standard”.

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. The approved algorithms for encryption and decryption algorithms are: the Advanced Encryption Standard (AES) and the Triple Data Encryption Algorithm (TDEA). TDEA is based on the Data Encryption Standard (DES), which is no longer approved for Federal Government use except as a component of TDEA. Each of these algorithms operates on blocks (chunks) of data during an encryption or decryption operation. For this reason, these algorithms are commonly referred to as block cipher algorithms.

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 E.2 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).



IEC 1141/13

Figure E.2 – 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 E.3.4).

E.3.3 Advanced Encryption Standard (AES)

NOTE 1 The following text is quoted from NIST SP 800-21:2005, 3.3.1.3.

The Advanced Encryption Standard (AES) was developed as a replacement for DES and is the preferred algorithm for new products. AES is specified in FIPS PUB 197. AES encrypts and decrypts data in 128-bit blocks, using 128, 192 or 256 bit keys. All three key sizes are adequate.

NOTE 2 The following text is quoted from RFC 5084.

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.

E.3.4 Encryption Modes of Operation

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.1.4.

With a symmetric key block cipher algorithm, the same plaintext block will always encrypt to the same ciphertext block when the same symmetric key is used. If the multiple blocks in a typical message (data stream) are encrypted separately, an adversary could easily substitute individual blocks, possibly without detection. Furthermore, certain kinds of data patterns in the plaintext, such as repeated blocks, would be apparent in the ciphertext.

Cryptographic modes of operation have been defined to address this problem by combining the basic cryptographic algorithm with variable initialization values (commonly known as initialization vectors) and feedback rules for the information derived from the cryptographic operation.

NIST SP 800-38D 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 5.4.8.

E.3.5 Message Authentication Code

E.3.5.1 General

NOTE The following text is quoted from NIST SP 800-21:2005, 3.3.2. In this context, “for Federal Government use” means “for the purposes of this standard”.

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 E.3 depicts the use of message authentication codes (MACs).

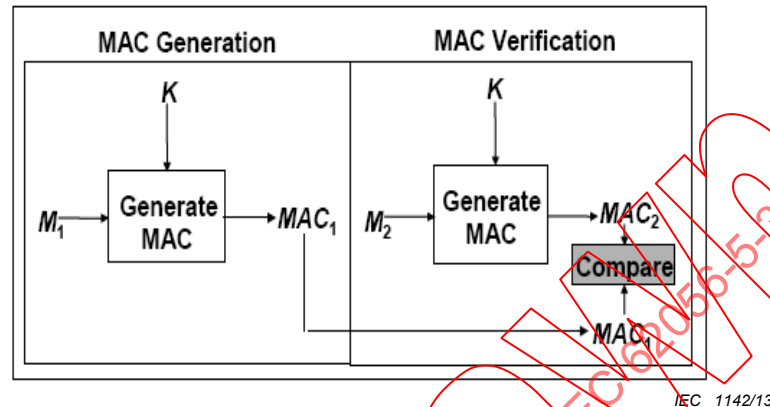


Figure E.3 – Message Authentication Codes (MACs)

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.

Two types of algorithms for computing a MAC have been approved for Federal government use: MAC algorithms that are based on block cipher algorithms, and MAC algorithms that are based on hash functions.

E.3.5.2 The Keyed-Hash Message Authentication Code (HMAC)

NOTE The following text is quoted from FIPS PUB 198.

The Keyed-Hash Message Authentication Code (HMAC) uses a cryptographic hash function in conjunction with a secret key. HMAC shall be used in combination with an approved cryptographic hash function. HMAC uses a secret key for the calculation and verification of the MACs. For details, see FIPS PUB 198.

E.3.6 Key establishment

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.

E.4 Asymmetric key algorithms

E.4.1 General

The use of asymmetric key algorithms for DLMS/COSEM is under consideration.

NOTE The following text is quoted from NIST SP 800-21:2005, 3.4.

Asymmetric 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 shall 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 encryption¹⁹, the encryption is performed using the public key, and the decryption is performed using the private key.

Asymmetric algorithms are used primarily as data integrity, authentication, and non-repudiation mechanisms (i.e., digital signatures), and for key establishment.

Some asymmetric 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 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 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.

E.4.2 Digital signatures

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

¹⁹ Not all public key algorithms are capable of multiple functions, e.g., generating digital signatures and encryption.

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 shall guard against the unauthorized acquisition of their private keys.

E.4.3 Key establishment

NOTE The following text is quoted from NIST SP 800-21:2005, 3.4.2.

Two types of asymmetric key (i.e., public key) establishment are defined: *key transport* and *key agreement*. Approved key establishment schemes are specified in NIST SP 800-56, *Recommendation on Key Establishment Schemes*.

Key transport is the distribution of a key (and other keying material) from one party to another party. The transported key is created by the sending party. The keying material is encrypted by the sending party and decrypted by the receiving party. The sending party encrypts the keying material using the receiving party's public key; the receiving party decrypts the received keying material using the associated private key.

Key agreement is the participation by both parties (i.e., the sending and receiving parties) in the creation of shared keying material. Each party has either one or two key pairs²⁰, and the public keys are made known to the other party. The key pairs are used to compute a shared secret²¹ value, which is then used with other information to derive keying material using a key derivation function. Typically, a hash function (see E.2) is used during the key derivation process.

²⁰ A key pair consists of a private key and its associated public key.

²¹ The shared secret is never transmitted from one party to the other.

Annex F (informative)

Significant technical changes with respect to IEC 62056-53

This part of the IEC 62056 includes the following significant technical changes with respect to IEC 62056-53:

- The title of this part has changed from “COSEM Application layer” to “DLMS/COSEM application layer”;
- Clause 2 includes a number of new references, mainly concerning data security;
- Clause 3 includes a number of new abbreviations, mainly concerning data security;
- Clause 4 of the previous edition, *The COSEM communications framework*, has been removed and will make the basis of the new Part IEC 62056-1-0;
- Subclause 4.2.3 has been re-organised. Each specific feature is now described in a separate subclause (derived from 7.4 of the previous edition);
- Subclause 4.2.3.1 *The xDLMS application service element* now contains Annex A of the previous edition, with necessary amendments;
- Clause 5 now specifies all elements concerning information security in DLMS/COSEM, including data access security – derived from IEC 62056-62:2006, 4.7 – and data transport security using ciphered xDLMS APDUs;
- Clause 6 has been reorganized: the four service primitives – .request, .indication, .response, .confirm as appropriate – of each service are described together. The service parameters are presented in a tabular form and they follow the order of the fields of the respective APDUs. In the description of the *Use of the services*, reference is made to the time sequence diagrams specified in 6.1;
- Subclause 6.1 now provides a general description of the service primitives and the notation used;
- Subclause 6.2, *The COSEM-OPEN service*: the Calling_AP_Title and the Responding_AP_Title parameters are now used in application contexts using ciphering;
- Subclause 6.3, *The COSEM-Release service*: the description has been amended for a better alignment with the ACSE A-Release service and to allow the user-information field of the RLRQ / RLRE to carry a ciphered xDLMS InitiateRequest / InitiateResponse APDU, to secure the COSEM-RELEASE service;
- Subclause 6.5, *Security parameters* contains the general specification of the security parameters used by data access services in ciphered application contexts;
- Subclauses 6.6 *The GET service*, 6.7 *The SET service* and 6.8 *The ACTION service*: the specification of the services using LN referencing has been amended for a coherent presentation with that of the services using SN referencing. The request and response primitive types are also presented in a table. These changes do not affect the protocol of the services or the encoding of the APDUs;
- Subclause 6.11 now includes the variable access specification parameter of the Read, Write and UnconfirmedWrite services used with SN referencing. New variants have been added to support block transfer;
- Subclause 0 now contains the detailed specification of the Read service. Block transfer is also available now. The use of the Variable_Access_Specification variants is presented in a table;
- Subclause 6.13 now contains the detailed specification of the Write service. Block transfer is also available now. The use of the Variable_Access_Specification variants is presented in a table;

- Subclause 6.14 now contains the detailed specification of the UnconfirmedWrite service. The use of the Variable_Access_Specification variants is presented in a table;
- Subclause 6.15 now contains the detailed specification of the InformationReport service;
- In 7.1.1, and 7.1.2, the state definitions are presented in a tabular form;
- Subclause 7.2.1 now includes the specification of the ACSE services. The description of the service parameters / fields and of the AARQ / AARE APDUs has been amended;
- Subclause 7.2.2.3, *The COSEM authentication mechanism name*: a new mechanism name (5) has been added for HLS with GMAC;
- Subclause 7.2.4.3 *Establishment of unconfirmed application associations* has been amended;
- Subclause 7.2.4.4 *Pre-established application associations* has been amended;
- Subclause 7.2.5 *Protocol for application association release* has been amended and now allows the use of a ciphered InitiateRequest APDUs in the user-information field;
- Subclause 7.3.1 now provides the description of the negotiation of services and options using the conformance block;
- Subclause 7.3.2 now summarizes all the rules concerning confirmed and unconfirmed service invocations;
- Subclause 7.3.3, *Protocol for the GET service*: the protocol specification and the figures have been amended. Transferring long data are also specified here. A new figure has been added. Table 36 shows the APDUs to be used with each request and response type. It is clarified, that:
 - if the value of a single attribute was requested, only the type and value shall be encoded (Data);
 - 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.

A new error case d) has been added: the block number sent by the server is not equal to the next in sequence,
- Subclause 7.3.4, *Protocol for the SET service*: the protocol specification and the figures have been amended in a similar way;
- Subclause 7.3.5, *Protocol for the ACTION service*, the protocol specification and the figures have been amended in a similar way;
- Subclause 7.3.6, *Protocol for the EventNotification service*: the specification has been amended;
- Subclause 7.3.7, *Protocol of the Read service*: the protocol specification has been amended to include block transfer. The mapping between the GET and ACTION services to the Read service has been brought and it is described in more detail;
- Subclause 7.3.8, *Protocol for the Write service*, the protocol specification has been amended to include block transfer. The mapping between the SET and ACTION services to the Write service has been brought and it is described in more detail;
- Subclause 7.3.9, *Protocol for the UnconfirmedWrite service*: the protocol of the UnconfirmedWrite service has been amended to cover block transfer. The mapping between the SET and ACTION services to the UnconfirmedWrite service has been brought and it is described in more detail;
- Subclause 7.3.10, *Protocol for the InformationReport service*: the protocol of the InformationReport service has been amended. The mapping between the EventNotification and the InformationReport service has been brought;
- Clause 8 *Abstract syntax*: The ASN.1 specification has been amended to include the changes for data security and SN block transfer. The elements have been re-arranged. The encoding of the user-information field tag [30] has been changed from IMPLICIT to EXPLICIT, to lead to constructed encoding, in line with the practice.

- Clause 8 *Abstract syntax*: The list of Data types has been modified. Added: time [11] // Tag of the “time” type in IEC 61334-4-41:1996, not usable in DLMS/COSEM. DLMS/COSEM uses [27]; UTF8-string [12] IMPLICIT OCTET-STRING // An ordered sequence of characters encoded as UTF-8. Deleted: floating-point [7] IMPLICIT OCTET STRING (SIZE(4)) // DLMS/COSEM uses types [23] and [24].
- Annex A includes now the former Clause B.1 of IEC 62056-53:2006;
- Clause B.2, *The 3-layer, connection-oriented, HDLC based communication profile* and Clause B.3 *The TCP-UDP/IP based communication profiles (COSEM_on_IP)* of IEC 62056-53:2006 have been removed and will appear in other parts of the IEC 62056 series;
- Annex B *AARQ and AARE encoding examples* includes the former Annex C of IEC 62056-53:2006. The examples have been amended. All examples are complete now and they are presented in a tabular form for easy comparison between the various cases;
- Annex C provides encoding examples of AARQ and AARE APDUs using a ciphered application context;
- Annex D includes data transfer service examples;
- Annex E provides an overview of cryptography.

Bibliography

DLMS UA 1000-1:2010, *COSEM Identification System and Interface Classes, the "Blue Book"*

DLMS UA 1000-2:2009, *DLMS/COSEM Architecture and Protocols, the "Green Book"*

DLMS UA 1001-1:2010, *DLMS/COSEM Conformance Test and certification process, the "Yellow Book"*

IEC 60050-300, *International Electrotechnical Vocabulary – Electrical and electronic measurements and measuring instruments –*

Part 311: General terms relating to measurements

Part 312: General terms relating to electrical measurements

Part 313: Types of electrical measuring instruments

Part 314: Specific terms according to the type of instrument

IEC 61334-4-32:1996, *Distribution automation using distribution line carrier systems – Part 4: Data communication protocols – Section 32: Data link layer – Logical link control (LLC)*

IEC 61334-4-511:2000, *Distribution automation using distribution line carrier systems – Part 4-511: Data communication protocols – Systems management – CIASE protocol*

IEC 61334-4-512:2001, *Distribution automation using distribution line carrier systems – Part 4-512: Data communication protocols – System management using profile 61334-5-1 – Management Information Base (MIB)*

IEC 61334-5-1:2001, *Distribution automation using distribution line carrier systems – Part 5-1: Lower layer profiles – The spread frequency shift keying (S-FSK) profile*

IEC/TR 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/TR 62056-51:1998, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 51: Application layer protocols*

IEC/TR 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 62056-1-0, *Electricity metering data exchange – Part 1-0: Smart metering standardization framework*²²

ISO/IEC 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*

ISO/IEC 8802-2:1998, *Information technology – Telecommunications and information exchange between systems – Local and metropolitan area networks – Specific requirements – Part 2: Logical link control*

²² Under consideration.

ISO/IEC 9545:1994, *Information technology – Open Systems Interconnection – Application layer structure*

ISO/IEC 10731:1994, *Information technology – Open Systems Interconnection – Basic Reference Model – Conventions for the definition of OSI services*

ISO/IEC 13239:2002, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*

ISO 2110:1989, *Information technology – Data communication – 25-pole DTE/DCE interface connector and contact number assignments*

ITU-T V.24:2000, *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 including procedures for disabling of echo control devices for both manually and automatically established calls*

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 for Low-Rate Wireless Personal Area Networks (WPANs)*

EN 13757-2:2004, *Communication system for and remote reading of meters – Part 2: Physical and Link Layer*

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 800-21:2005, *Guideline for Implementing Cryptography in the Federal Government*

RFC 5084:2007, Internet Engineering Task Force (IETF). *Using AES-CCM and AES-GCM Authenticated Encryption in the Cryptographic Message Syntax (CMS)*. Edited by R. Housley November 2007. Available from: <http://www.rfc-editor.org/rfc/rfc5084.txt>

McGrew D.A. and Viega J., *The Galois/Counter Mode of Operation (GCM)*:2005. Available from: Cisco Systems, Inc. 170, West Tasman Drive, San Jose, CA 95032, mcgrew@cisco.com and Secure Software, 4100 Lafayette Center Drive, Suite 100, Chantilly, VA 20151, viega@securesoftware.com

ANSI C12.21:1999, *Protocol Specification for Telephone Modem Communication*

Index

- AA, confirmed, 14
- AA, pre-established, 15
- AA, unconfirmed, 14
- AARQ and AARE encoding examples, 131
- A-ASSOCIATE service, 14, 43
- Abstract syntax, 20
- Abstract syntax, COSEM APDUs, 115
- Access right, 21
- Access_Selection_Parameters, 53, 56
- ACSE functional units, 78
- ACSE procedures, 20
- ACSE protocol version, 45, 87
- ACSE requirements, 80
- ACSE services and APDUs, 78
- ACTION service, 17, 58, 101
- ACTION.confirm, 62
- ACTION.indication, 62
- ACTION.request, 62
- ACTION.response, 62
- Action-Request, 62
- ACTION-REQUEST-FIRST-BLOCK, 60, 102, 105, 110
- ACTION-REQUEST-LAST-BLOCK, 60, 102, 106, 110
- ACTION-REQUEST-NEXT, 60, 102, 105
- Action-Request-Next-Pblock, 102
- Action-Request-Normal, 102
- ACTION-REQUEST-NORMAL, 60, 102, 105, 110, 113
- ACTION-REQUEST-ONE-BLOCK, 60, 102, 106, 110
- Action-Request-With-First-Pblock, 102
- Action-Request-With-List, 102
- ACTION-REQUEST-WITH-LIST, 60, 102, 106, 110, 113
- ACTION-REQUEST-WITH-LIST-AND-FIRST-BLOCK, 60, 61, 102, 106, 110
- Action-Request-With-List-And-With-Frist-Pblock, 102
- Action-Request-With-Pblock, 102
- Action-Response, 62
- ACTION-RESPONSE-LAST-BLOCK, 60, 102, 106
- ACTION-RESPONSE-NEXT, 60, 102, 106, 110
- Action-Response-Next-Pblock, 102
- Action-Response-Normal, 102
- ACTION-RESPONSE-NORMAL, 60, 102, 106, 110
- ACTION-RESPONSE-ONE-BLOCK, 60, 102
- ACTION-RESPONSE-ONE-ONE-BLOCK, 106
- Action-Response-With-List, 102
- ACTION-RESPONSE-WITH-LIST, 60, 102, 106, 110
- Action-Response-With-Pblock, 102
- Additional authenticated data, 33, 34, 35
- Additional data, 31
- Advanced Encryption Standard, 31, 169, 170
- AES-128 key wrap algorithm, 26, 30
- AL service, non-client/server type, 16
- AL services, AA establishment and release, 14
- AL services, client/server type, 16
- AL services, data transfer, 15
- AL, management services, 74
- Application association, 13, 48
- Application association, confirmed, 83
- Application association, establishment, 83
- Application association, graceful release, 88
- Application association, non-graceful release, 91
- Application association, pre-established, 88
- Application association, release, 88
- Application association, unconfirmed, 87
- Application context, 20
- Application context name, 14, 45, 80, 81, 86
- Application Control Service Element, 13
- Application Programming Interface, 16
- Application_Addresses parameter, 63
- A-RELEASE, 48
- A-RELEASE service, 15
- ASN.1, 81
- Association LN, 21
- Association SN, 21
- Attribute_0 referencing, 18
- Authenticated decryption, 33, 34
- Authenticated encryption, 31, 33
- Authentication, 20, 26, 168, 172
- Authentication context, 20
- Authentication key, 26, 32, 33, 35, 39
- Authentication mechanism name, 14, 82
- Authentication tag, 24, 27, 28, 32, 33, 34, 35, 39, 52
- Authentication value, 80
- authentication_mechanism_id(5), 39
- Authenticity, 169
- Authorization, 168
- A-XDR encoding, 14
- BER encoding, 14
- Block cipher, 32
- Block cipher algorithm, 169
- Block cipher key, 26, 33, 35
- Block_Number, 54, 57, 61, 67, 68, 71, 96, 100
- Block_Number_Access, 64, 67
- Broadcast, 28, 37
- Calling authentication value, 22, 23, 46
- Central DCS, 29
- Challenge, 22
- Ciphered ADPUs, 20
- Ciphered xDLMS APDU, 27
- Ciphertext, 28, 32, 33, 34, 169
- Client side layer management services, 74
- Client SN_MAPPER, 17
- Client system title, 86
- Client_Max_Receive_PDU_Size, 46
- client_system_title, 37
- client-max-receive-pdu-size, 86
- Communication environment, 129
- Communication profile, 9
- Communication profile specific parameters, 130
- Communication profile structure, 129
- Concentrator, 29
- Confidentiality, 20, 28, 168, 169
- ConfirmedServiceError, 26, 68, 71, 93
- Conformance block, 17, 92
- Control function, 13, 75
- COSEM AL, ASO services, 14
- COSEM AL, layer management services, 19
- COSEM AL, protocol specification, 20
- COSEM AL, service specification, 41
- COSEM AL, services, 14
- COSEM application context, 17, 27
- COSEM application context name, 81
- COSEM application layer, protocol specification, 75
- COSEM client/server type services, 17
- COSEM interface object, 13, 16
- COSEM_Application_Context_Name, 82
- COSEM_Attribute_Descriptor, 53, 56
- COSEM_Authentication_Mechanism_Name, 82
- COSEM_Class_Id, 53, 56, 60
- COSEM_Method_Descriptor, 60
- COSEM_Method_Id, 60
- COSEM_Object_Attribute_Id, 53, 56
- COSEM_Object_Instance_Id, 53, 56, 60
- COSEM-ABORT service, 15, 50
- COSEM-OPEN service, 14, 22, 43
- COSEM-OPEN service invocations, repeated, 87
- COSEM-RELEASE service, 15, 48
- Counter mode, 31
- Cryptographic keys, 28
- Cryptography, 168
- Data access security, 20
- Data integrity, 168, 172
- Data transfer services, protocol, 92
- Data transport security, 20
- Data_Access_Error, 68, 71
- Data_Access_Result, 57
- DataBlock_G, 54, 96, 97
- DataBlock_SA, 57, 61, 100
- Decryption, 169
- Dedicated key, 29, 30, 46
- Definitions, 11
- Denial-of-service attack, 49

Deterministic construction, 35
 Digital signature, 172, 173
 DLMS conformance, 46
 DLMS version number, 46
 Eavesdropping, 21
 Encryption, 26, 169
 Event notification, 17
 EventNotification service, 17, 62
 ExceptionResponse, 26, 94
 Failure_type, 45
 Fixed field, 36
 Forward cipher function, 33
 Frame counter, 27
 Galois/Counter Mode, 26, 31, 170
 GET service, 17, 52, 94
 GET.confirm, 55
 GET.indication, 54
 GET.request, 54
 GET.response, 54
 Get_Data_Result, 54
 Get-Request, 54
 Get-Request-Next, 94
 GET-REQUEST-NEXT, 53, 94, 97, 105
 Get-Request-Normal, 94
 GET-REQUEST-NORMAL, 53, 94, 105
 Get-Request-With-List, 95
 GET-REQUEST-WITH-LIST, 53, 95, 105
 Get-Response, 55
 GET-RESPONSE-LAST-BLOCK, 53, 95, 105
 Get-Response-Normal, 94
 GET-RESPONSE-NORMAL, 53, 94, 105
 GET-RESPONSE-ONE-BLOCK, 53, 94, 97, 105
 Get-Response-With-Datablock, 94
 Get-Response-With-List, 95
 GET-RESPONSE-WITH-LIST, 53, 95, 105
 Global key, 29
 Global unicast encryption key, 39
 global_key_transfer, 30
 Hash value, 168
 High level security, 22, 39
 HLS secret, 39
 Identification and addressing scheme, 129
 Identifying service invocations, 17
 Implementation information, 46, 80
 InformationReport service, 17, 73, 113
 InformationReport.request, 113
 InformationReportRequest, 113
 Initialization vector, 26, 32, 33, 34, 35, 170
 Integrity, 20, 169
 Intrinsic security, 22
 Invocation field, 32, 36, 37
 Invoke_Id, 52, 55, 59, 97, 101, 103
 Invoke_Id parameter, 47
 Key agreement, 173
 Key encrypting keys, 28
 Key establishment, 172, 173
 Key management, 29
 Key transport, 173
 Key wrapping key, 28, 172
 Key_set, 27, 35
 Keyed-Hash Message Authentication Code, 171
 Last_Block, 54, 57, 61, 67, 68, 71, 96, 100
 LN referencing, 16
 LN/SN data transfer service mapping, 75
 Local_or_Remote, 45, 49
 Logical Name, 14
 Long service parameters, 19
 Low level security, 21
 Lowest level security, 21
 Manufacturer ID, 36
 Manufacturing number, 36
 Master key, 29
 MD5 algorithm, 22
 Mechanism name, 80
 Message Authentication Code, 171
 Message digest, 168
 Message integrity, 28, 171
 Message replay, 21
 Message source, 28
 Mode of Operation, 170
 Multiple references, 18
 NIST, 20
 Nonce, 32, 33
 Non-repudiation, 168, 172
 Parameterized_Access, 64, 66, 67, 70, 71, 73
 Plaintext, 28, 32, 33, 34, 35, 169
 Pre-established application association, 47
 Presentation layer, 14
 Priority, 53, 55, 59, 97, 101, 103
 Priority parameter, 18
 Private key, 28, 172
 proposed-conformance, 86
 proposed-dlms-version-number, 86
 Protocol connection parameters, 45
 Protocol version, 80
 Public key, 28, 172
 Raw_Data, 54, 57, 61, 67, 68, 100
 Read service, 17, 65, 104
 Read.confirm, 69
 Read.indication, 69
 Read.request, 68
 Read.response, 69
 Read_Data_Block_Access, 64, 67
 ReadRequest, 68, 105
 ReadResponse, 68, 69, 105, 106
 Reason, 81
 Referencing method, 13
 Registered COSEM names, 81
 reply_to_HLS_authentication, 23
 Request_Type, 53, 56, 59
 Responding-AP title, 86
 Response_Type, 53, 56
 response_allowed, 83, 86, 87
 Result, 45, 80, 96
 Result (–), 68, 71
 Result (+), 67, 71
 Result Source-Diagnostic, 80
 RLRE APDU, 48
 RLRQ APDU, 48
 Secret, 22
 Secret key, 22
 Security attributes, 25
 Security context, 25
 Security control, 27, 35, 51
 Security header, 27, 35
 Security material, 24
 Security mechanism name, 23, 46
 Security policy, 25
 Security setup, 30
 Security suite, 26
 Security_Options, 24, 51
 Security_Status, 24, 51
 Selective access, 18
 Sender ACSE requirements, 86
 Server system title, 86
 Server_Max_Receive_PDU_Size, 46
 server_system_title, 37
 Service_Class, 53, 56, 59
 Service_Class == Unconfirmed, 47
 Service_Class parameter, 47
 SET service, 17, 55, 98
 SET.confirm, 58
 SET.indication, 58
 SET.request, 57
 SET.response, 58
 SetMapperTables.request, 74
 Set-Request, 57
 SET-REQUEST-FIRST-BLOCK, 56, 99, 109
 SET-REQUEST-FIRST-BLOCK-WITH-LIST, 56, 57, 99, 109
 SET-REQUEST-LAST-BLOCK, 56, 99, 109
 Set-Request-Normal, 99
 SET-REQUEST-NORMAL, 56, 57, 99, 109, 112
 SET-REQUEST-ONE-BLOCK, 56, 99, 109

Set-Request-With-Datablock, 99
 SET-REQUEST-WITH-FIRST-BLOCK, 57
 Set-Request-With-First-Datablock, 99
 Set-Request-With-List, 99
 SET-REQUEST-WITH-LIST, 56, 57, 99, 109, 112
 Set-Request-With-List-And-With-First-Datablock, 99
 Set-Response, 58
 SET-RESPONSE-ACK-BLOCK, 56, 99, 109
 Set-Response-Datablock, 99
 SET-RESPONSE-LAST-BLOCK, 56, 99, 109
 SET-RESPONSE-LAST-BLOCK-WITH-LIST, 56, 99, 109
 Set-Response-Last-Datablock, 99
 Set-Response-Normal, 99
 SET-RESPONSE-NORMAL, 56, 99, 109
 Set-Response-With-List, 99
 SET-RESPONSE-WITH-LIST, 56, 99, 109
 S-FSK PLC environment, 36
 SHA-1 algorithm, 22
 Short Name, 14
 SN referencing, 16
 SN_MAPPER ASE, 68, 72, 73
 Supporting layer services and service mapping, 130
 Symmetric key, 28
 Symmetric key algorithm, 169
 Symmetric key block cipher, 31
 System title, 36
 TriggerEventNotificationSending service, 63
 UnconfirmedWrite service, 17, 72, 112
 UnconfirmedWrite.indication, 73
 UnconfirmedWrite.request, 73
 UnconfirmedWriteRequest, 73
 Unicast, 28
 User information, 47, 81
 Variable Access Specification, 64
 Variable_Access_Specification, 67
 Variable_Name, 64, 66, 67, 70, 71, 73
 Variable-Access-Specification, 66, 70, 73
 Write service, 17, 69, 108
 Write.confirm, 72
 Write.indication, 72
 Write.request, 72
 Write.response, 72
 Write_Data_Block_Access, 64, 70
 WriteRequest, 72, 109
 WriteResponse, 71, 72, 109
 xDLMS context, 14
 xDLMS InitiateRequest, 30
 xDLMS InitiateResponse, 30
 xDLMS procedures, 20
 xDLMS_ASE, 13

Withd
 IECNORM.COM: Click to view the full PDF of IEC 62056-5-3:2013

IECNORM.COM: Click to view the full PDF of IEC 62056-5-3:2013

SOMMAIRE

AVANT-PROPOS	183
1 Domaine d'application	185
2 Références normatives	185
3 Termes, définitions et abréviations	187
3.1 Termes et définitions	187
3.2 Abréviations	187
4 Vue d'ensemble	189
4.1 Structure de la couche application COSEM	189
4.2 Services de la couche application COSEM	190
4.2.1 Services ASO	190
4.2.2 Services fournis pour l'établissement et la libération d'associations d'applications	190
4.2.3 Services fournis pour le transfert de données	191
4.2.4 Services de gestion de couche	195
4.2.5 Récapitulatif des services de la couche application COSEM	195
4.3 Protocoles de la couche application COSEM	196
5 Sécurité des informations dans DLMS/COSEM	196
5.1 Définitions	197
5.2 Généralités	197
5.3 Sécurité de l'accès aux données	198
5.3.1 Vue d'ensemble	198
5.3.2 Niveau de sécurité le plus faible (pas de sécurité)	198
5.3.3 Niveau de sécurité faible (LLS)	198
5.3.4 Niveau de Sécurité élevé (HLS)	199
5.4 Sécurité de transport des données	201
5.4.1 Application, suppression ou vérification de la protection: chiffrement et déchiffrement	201
5.4.2 Contexte de sécurité	203
5.4.3 Politique de sécurité	203
5.4.4 Suite de sécurité	204
5.4.5 Matériel de sécurité	204
5.4.6 APDU xDLMS chiffrées	204
5.4.7 Clés cryptographiques	206
5.4.8 Mode de fonctionnement Galois/Counter (GCM)	209
6 Spécification de service de la couche application COSEM	220
6.1 Primitives de service et paramètres	220
6.2 Service COSEM-OPEN	224
6.3 Service COSEM-RELEASE	228
6.4 Service COSEM-ABORT	231
6.5 Paramètres de sécurité	231
6.6 Service GET	232
6.7 Service SET	235
6.8 Service ACTION	238
6.9 Service EventNotification	242
6.10 Service TriggerEventNotificationSending	243
6.11 Spécification d'accès variable	244

6.12	Service Read.....	245
6.13	Service Write.....	249
6.14	Service UnconfirmedWrite	252
6.15	Service InformationReport	254
6.16	Services de gestion de couches côté client: Demande SetMapperTable.request	255
6.17	Récapitulatif des services et du mappage de service de transfert de données LN/SN	255
7	Spécification du protocole de couche application COSEM	256
7.1	Fonction de commande	256
7.1.1	Définitions des états de la fonction de commande côté client	256
7.1.2	Définitions des états de la fonction de commande côté serveur	258
7.2	Services ACSE et APDU	259
7.2.1	Unités fonctionnelles ACSE, services et paramètres de service	259
7.2.2	Noms COSEM enregistrés	262
7.2.3	Règles de codage d'APDU.....	264
7.2.4	Protocole d'établissement d'association d'applications	264
7.2.5	Protocole de libération d'association d'applications	270
7.3	Protocole des services de transfert de données	276
7.3.1	Négociation de services et d'options - Bloc de conformité.....	276
7.3.2	Appels de service confirmés et non confirmés	277
7.3.3	Protocole du service GET	278
7.3.4	Protocole du service SET	283
7.3.5	Protocole du service ACTION	286
7.3.6	Protocole du service EventNotification.....	289
7.3.7	Protocole du service Read.....	289
7.3.8	Protocole du service Write.....	293
7.3.9	Protocole du service UnconfirmedWrite	297
7.3.10	Protocole du service InformationReport	298
8	Syntaxe abstraite des APDU ACSE et COSEM.....	299
	Annexe A (normative) Utilisation de la couche application COSEM dans différents profils de communication	313
	Annexe B (informative) Exemples de codage AARQ et AARE	315
	Annexe C (informative) Exemples de codage: APDU AARQ et AARE utilisant un contexte d'application chiffré.....	328
	Annexe D (informative) Exemples de services de transfert de données.....	337
	Annexe E (informative) Présentation de la cryptographie	353
	Annexe F (informative) Modifications techniques majeures par rapport à la CEI 62056- 53	360
	Bibliographie.....	363
	Index	366
	Figure 1 – Structure des couches application COSEM	189
	Figure 2 – Récapitulatif des services de l'AL COSEM	196
	Figure 3 – Authentification LLS et HLS	201
	Figure 4 – Sécurité de transport des données dans DLMS/COSEM.....	203
	Figure 5 – APDU xDLMS chiffrées	205
	Figure 6 – Protection cryptographique des APDU xDLMS à l'aide de GCM	215

Figure 7 – Primitives de service	220
Figure 8 – Diagrammes de séquences temporelles	222
Figure 9 – Machine à état, partielle, pour la fonction de commande côté client	257
Figure 10 – État partiel de la machine pour la fonction de commande côté serveur	258
Figure 11 – MSC pour l'établissement réussi d'une AA précédé de l'établissement réussi d'une connexion de couche inférieure de support	267
Figure 12 – Libération d'AA sans perte de données à l'aide du service A-RELEASE	272
Figure 13 – Libération d'AA sans perte de données par déconnexion de la couche de support	274
Figure 14 – Abandon d'une AA suite à la primitive PH-ABORT.indication	276
Figure 15 – MSC du service GET	279
Figure 16 – MSC du service GET avec transfert de bloc	280
Figure 17 – MSC du service GET avec transfert de bloc, GET long abandonné	283
Figure 18 – MSC du service SET	284
Figure 19 – MSC du service SET avec transfert de bloc	285
Figure 20 – MSC du service ACTION	287
Figure 21 – MSC du service ACTION avec transfert de bloc	288
Figure 22 – MSC du service Read utilisé pour lire un attribut	292
Figure 23 – MSC du service Read utilisé pour appeler une méthode	292
Figure 24 – MSC du service Read utilisé pour lire un attribut, avec transfert de blocs	293
Figure 25 – MSC du service Write utilisé pour écrire un attribut	296
Figure 26 – MSC du service Write utilisé pour appeler une méthode	296
Figure 27 – MSC du service Write utilisé pour écrire un attribut, avec transfert de blocs	297
Figure 28 – MSC du service Unconfirmed Write utilisé pour écrire un attribut	298
Figure E.1 – Fonction de hachage	354
Figure E.2 – Chiffrement et déchiffrement	355
Figure E.3 – Codes d'authentification de message (MAC)	356
Tableau 1 – Explication de la signification des paramètres PDU Size pour DLMS/COSEM	192
Tableau 2 – Suites de sécurité	204
Tableau 3 – Champ Security Control	206
Tableau 4 – Clés cryptographiques et leur gestion	209
Tableau 5 – Texte brut et AAD	215
Tableau 6 – Exemple d'APDU chiffrées	218
Tableau 7 – Exemple HLS avec GMAC	219
Tableau 8 – Codes des paramètres de service de l'AL	223
Tableau 9 – Paramètres de service des primitives de service COSEM-OPEN	224
Tableau 10 – Paramètres de service des primitives de service COSEM-RELEASE	228
Tableau 11 – Paramètres de service des primitives de service COSEM-ABORT	231
Tableau 12 – Paramètres de sécurité	232
Tableau 13 – Paramètres de service du service GET	233
Tableau 14 – Types de demande et de réponse du service GET	234

Tableau 15 – Paramètres du service SET	236
Tableau 16 – Types de demande et de réponse du service SET	237
Tableau 17 – Paramètres du service ACTION	239
Tableau 18 – Types de demande et de réponse du service ACTION	240
Tableau 19 – Paramètres de service des primitives du service EventNotification	243
Tableau 20 – Paramètres de service de la primitive de service TriggerEventNotificationSending.request	244
Tableau 21 – Spécification d'accès variable	245
Tableau 22 – Paramètres du service Read	246
Tableau 23 – Utilisation des variantes du paramètre Variable_Access_Specification et des choix de Read.response	247
Tableau 24 – Paramètres du service Write	250
Tableau 25 – Utilisation des variantes de Variable_Access_Specification et des choix de Write.response	251
Tableau 26 – Paramètres du service UnconfirmedWrite	253
Tableau 27 – Utilisation des variantes de Variable_Access_Specification	253
Tableau 28 – Paramètres du service InformationReport	254
Tableau 29 – Paramètres de service des primitives de service SetMapperTable.request	255
Tableau 30 – Récapitulatif des services ACSE	255
Tableau 31 – Récapitulatif des services xDLMS pour le référencement LN	256
Tableau 32 – Récapitulatif des services xDLMS pour le référencement SN	256
Tableau 33 – Unités fonctionnelles ACSE, services et paramètres de service	260
Tableau 34 – Utilisation des APDU chiffrées et non chiffrées	263
Tableau 35 – Bloc de conformité xDLMS	276
Tableau 36 – Types et APDU de service GET	278
Tableau 37 – Types et APDU de service SET	283
Tableau 38 – Types et APDU de service ACTION	286
Tableau 39 – Mappage entre le service GET et le service Read	290
Tableau 40 – Mappage entre le service ACTION et le service Read	290
Tableau 41 – Mappage entre le service SET et le service Write	294
Tableau 42 – Mappage entre le service ACTION et le service Write	295
Tableau 43 – Mappage entre le service SET et le service UnconfirmedWrite	298
Tableau 44 – Mappage entre le service ACTION et le service UnconfirmedWrite	298
Tableau 45 – Mappage entre les services EventNotification et InformationReport	299
Tableau B.1 – Bloc de conformité	316
Tableau B.3 – Codage A-XDR de l'APDU xDLMS InitiateResponse	318
Tableau B.4 – Codage BER de l'APDU AARQ	321
Tableau B.5 – APDU AARQ complète	323
Tableau B.6 – Codage BER de l'APDU AARE	324
Tableau B.7 – APDU AARE complète	327
Tableau C.1 – Codage A-XDR de l'APDU xDLMS InitiateRequest	328
Tableau C.2 – Chiffrement authentifié de l'APDU xDLMS InitiateRequest	329
Tableau C.3 – Codage BER de l'APDU AARQ	330
Tableau C.4 – Codage A-XDR de l'APDU xDLMS InitiateResponse	332

Tableau C.5 – Chiffrement authentifié de l'APDU xDLMS InitiateResponse	332
Tableau C.6 – Codage BER de l'APDU AARE	333
Tableau C.7 – Codage BER de l'APDU RLRQ	335
Tableau C.8 – Codage BER de l'APDU RLRE	336
Tableau D.1 – Objets utilisés dans les exemples	337
Tableau D.2 – Exemple: Lecture de la valeur d'un attribut unique sans transfert de bloc.....	338
Tableau D.3 – Exemple: Lecture de la valeur d'une liste d'attributs sans transfert de bloc	339
Tableau D.4 – Exemple: Lecture d'un attribut unique avec transfert de bloc.....	340
Tableau D.5 – Exemple: Lecture de la valeur d'une liste d'attributs avec transfert de bloc	343
Tableau D.6 – Exemple: Écriture de la valeur d'un attribut unique sans transfert de bloc	346
Tableau D.7 – Exemple: Écriture de la valeur d'une liste d'attributs sans transfert de bloc	347
Tableau D.8 – Exemple: Écriture de la valeur d'un attribut unique avec transfert de bloc	348
Tableau D.9 – Exemple: Écriture de la valeur d'une liste d'attributs avec transfert de bloc	350

IECNORM.COM: Click to view the full PDF of IEC 62056-5-3:2013

COMMISSION ÉLECTROTECHNIQUE INTERNATIONALE

**ÉCHANGE DES DONNÉES
DE COMPTAGE DE L'ÉLECTRICITÉ –
LA SUITE DLMS/COSEM –****Partie 5-3: Couche application DLMS/COSEM****AVANT-PROPOS**

- 1) La Commission Electrotechnique Internationale (CEI) est une organisation mondiale de normalisation composée de l'ensemble des comités électrotechniques nationaux (Comités nationaux de la CEI). La CEI a pour objet de favoriser la coopération internationale pour toutes les questions de normalisation dans les domaines de l'électricité et de l'électronique. A cet effet, la CEI – entre autres activités – publie des Normes internationales, des Spécifications techniques, des Rapports techniques, des Spécifications accessibles au public (PAS) et des Guides (ci-après dénommés "Publication(s) de la CEI"). Leur élaboration est confiée à des comités d'études, aux travaux desquels tout Comité national intéressé par le sujet traité peut participer. Les organisations internationales, gouvernementales et non gouvernementales, en liaison avec la CEI, participent également aux travaux. La CEI collabore étroitement avec l'Organisation Internationale de Normalisation (ISO), selon des conditions fixées par accord entre les deux organisations.
- 2) Les décisions ou accords officiels de la CEI concernant les questions techniques représentent, dans la mesure du possible, un accord international sur les sujets étudiés, étant donné que les Comités nationaux de la CEI intéressés sont représentés dans chaque comité d'études.
- 3) Les Publications de la CEI se présentent sous la forme de recommandations internationales et sont agréées comme telles par les Comités nationaux de la CEI. Tous les efforts raisonnables sont entrepris afin que la CEI s'assure de l'exactitude du contenu technique de ses publications, la CEI ne peut pas être tenue responsable de l'éventuelle mauvaise utilisation ou interprétation qui en est faite par un quelconque utilisateur final.
- 4) Dans le but d'encourager l'uniformité internationale, les Comités nationaux de la CEI s'engagent, dans toute la mesure possible, à appliquer de façon transparente, les Publications de la CEI dans leurs publications nationales et régionales. Toutes divergences entre toutes Publications de la CEI et toutes publications nationales ou régionales correspondantes doivent être indiquées en termes clairs dans ces dernières.
- 5) La CEI elle-même ne fournit aucune attestation de conformité. Des organismes de certification indépendants fournissent des services d'évaluation de conformité et, dans certains secteurs, accèdent aux marques de conformité de la CEI. La CEI n'est responsable d'aucun des services effectués par les organismes de certification indépendants.
- 6) Tous les utilisateurs doivent s'assurer qu'ils sont en possession de la dernière édition de cette publication.
- 7) Aucune responsabilité ne doit être imputée à la CEI, à ses administrateurs, employés, auxiliaires ou mandataires, y compris ses experts particuliers et les membres de ses comités d'études et des Comités nationaux de la CEI, pour tout préjudice causé en cas de dommages corporels et matériels, ou de tout autre dommage de quelque nature que ce soit, directe ou indirecte, ou pour supporter les coûts (y compris les frais de justice) et les dépenses découlant de la publication ou de l'utilisation de cette Publication de la CEI ou de toute autre Publication de la CEI, ou au crédit qui lui est accordé.
- 8) L'attention est attirée sur les références normatives citées dans cette publication. L'utilisation de publications référencées est obligatoire pour une application correcte de la présente publication.
- 9) L'attention est attirée sur le fait que certains des éléments de la présente Publication de la CEI peuvent faire l'objet de droits de brevet. La CEI ne saurait être tenue pour responsable de ne pas avoir identifié de tels droits de brevets et de ne pas avoir signalé leur existence.

La Commission Electrotechnique Internationale (CEI) attire l'attention sur le fait qu'il est déclaré que la conformité à la présente Norme internationale peut nécessiter l'utilisation d'un service de maintenance concernant la pile de protocoles sur laquelle est basée la présente Norme CEI 62056-5-3.

La CEI ne prend pas position concernant la preuve, la validité et le domaine d'application de ce service de maintenance.

Le fournisseur du service de maintenance a assuré à la CEI qu'il souhaite fournir des services aux demandeurs dans le monde entier, selon des termes et les conditions raisonnables et non discriminatoires. À cet égard, la déclaration du fournisseur du service de maintenance est enregistrée avec la CEI. Des informations peuvent être obtenues auprès de:

DLMS¹ User Association
Zug/Switzerland
www.dlms.ch

La Norme internationale CEI 62056-5-3 a été établie par le comité d'études 13 de la CEI: Mesure de l'énergie électrique, contrôle des tarifs et de la charge.

Cette édition annule et remplace la CEI 62056-53 parue en 2006. Cette édition constitue une révision technique.

Les modifications techniques majeures par rapport à la CEI 62056-53 sont énumérées dans l'Annexe F.

Le texte de cette norme est issu des documents suivants:

FDIS	Rapport de vote
13/1523/FDIS	13/1541/RVD

Le rapport de vote indiqué dans le tableau ci-dessus donne toute information sur le vote ayant abouti à l'approbation de cette norme.

Cette publication a été rédigée selon les Directives ISO/CEI, Partie 2.

Une liste de toutes les parties de la série CEI 62056, publiées sous le titre général *Échange des données de comptage de l'électricité – La suite DLMS/COSEM*, peut être consultée sur le site web de la CEI.

Les futures normes de cette série porteront dorénavant le nouveau titre général cité ci-dessus. Le titre des normes existant déjà dans cette série sera mis à jour lors de la prochaine édition.

La numérotation est passée de CEI 62056-XY à CEI 62056-X-Y. Par exemple, la CEI 62056-53 devient la CEI 62056-5-3.

Le comité a décidé que le contenu de cette publication ne sera pas modifié avant la date de stabilité indiquée sur le site web de la CEI sous "<http://webstore.iec.ch>" dans les données relatives à la publication recherchée. A cette date, la publication sera

- reconduite,
- supprimée,
- remplacée par une édition révisée, ou
- amendée.

IMPORTANT – Le logo "colour inside" qui se trouve sur la page de couverture de cette publication indique qu'elle contient des couleurs qui sont considérées comme utiles à une bonne compréhension de son contenu. Les utilisateurs devraient, par conséquent, imprimer cette publication en utilisant une imprimante couleur.

¹ Spécification de message de langage de dispositif.

ÉCHANGE DES DONNÉES DE COMPTAGE DE L'ÉLECTRICITÉ – LA SUITE DLMS/COSEM –

Partie 5-3: Couche application DLMS/COSEM

1 Domaine d'application

La présente partie de la CEI 62056 indique la couche application DLMS/COSEM en termes de structure, de services et de protocoles pour les clients et serveurs COSEM, et définit comment utiliser la couche application DLMS/COSEM dans différents profils de communication.

Elle définit les services permettant d'établir et de libérer des associations d'applications, ainsi que les services de communication de données permettant d'accéder aux méthodes et aux attributs des objets d'interface COSEM, définis dans la CEI 62056-6-2², à l'aide du référencement par nom logique (LN) ou par nom abrégé (SN).

L'Annexe A (normative) définit comment utiliser la couche application COSEM dans différents profils de communication. Elle indique comment différents profils de communication peuvent être construits de sorte à échanger des données avec les équipements de mesure à l'aide du modèle d'interface COSEM, ainsi que les éléments nécessaires à indiquer dans chaque profil de communication. Les profils de communication réels, spécifiques au support, sont spécifiés dans des parties distinctes de la série CEI 62056.

L'Annexe B, l'Annexe C et l'Annexe D (informatives) incluent des exemples de codage d'APDU.

L'Annexe E (informative) présente la cryptographie.

2 Références normatives

Les documents suivants sont cités en référence de manière normative, en intégralité ou en partie, dans le présent document et sont indispensables pour son application. Pour les références datées, seule l'édition citée s'applique. Pour les références non datées, la dernière édition du document de référence s'applique (y compris les éventuels amendements).

CEI 61334-4-41:1996, *Automatisation de la distribution à l'aide de systèmes de communication à courants porteurs – Partie 4: Protocoles de communication de données – Section 41: Protocoles d'application – Spécification des messages de ligne de distribution*

CEI 61334-6:2000, *Automatisation de la distribution à l'aide de systèmes de communication à courants porteurs – Partie 6: Règles d'encodage A-XDR*

CEI/TR 62051:1999, *Lecture des compteurs électriques – Glossaire de termes* (disponible en anglais seulement)

² A publier simultanément avec la présente partie de la CEI 62056.

CEI/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* (disponible en anglais seulement)

CEI 62056-6-1:—, *Échange des données de comptage de l'électricité – La suite DLMS/COSEM – Partie 6-1: Système d'identification des objets (OBIS)*³

CEI 62056-6-2:—, *Échange des données de comptage de l'électricité – La suite DLMS/COSEM – Partie 6-2: Classes d'interface COSEM*⁴

CEI 62056-8-3:—, *Échange des données de comptage de l'électricité – La suite DLMS/COSEM – Partie 8-3: Profil de communication pour réseaux de voisinage CPL S-FSK*⁵

ISO/CEI 15953:1999, *Technologies de l'information – Interconnexion des systèmes ouverts – Définition du service pour l'élément de service de contrôle d'association des objets de service d'application*

ISO/CEI 15954:1999, *Technologies de l'information – Interconnexion des systèmes ouverts – Protocole en mode connexion pour l'élément de service de contrôle d'association des objets de service d'application*

ISO/CEI 8824-1:2008, *Technologies de l'information – Notation de syntaxe abstraite numéro un (ASN.1): Spécification de la notation de base* (disponible en anglais seulement)

ISO/CEI 8825-1:2008, *Technologies de l'information – Règles de codage ASN.1: Spécification des règles de codage de base (BER), des règles de codage canoniques (CER) et des règles de codage distinctives (DER)* (disponible en anglais seulement)

FIPS PUB 180-1:2002, *Secure hash standard* (disponible en anglais seulement)

FIPS PUB 197:2001, *Advanced Encryption Standard (AES)* (disponible en anglais seulement)

NIST SP 800-38D:2007, *Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC* (disponible en anglais seulement)

NIST SP 800-57:2006, *Recommendation for Key Management – Part 1: General* (Revised) (disponible en anglais seulement)

RFC 1321:1992, Internet Engineering Task Force (IETF). *The MD5 Message-Digest Algorithm*. Édité par R. Rivest (MIT Laboratory for Computer Science and RSA Data Security, Inc.) avril 1992. Disponible à l'adresse: <http://www.rfc-editor.org/rfc/rfc1321.txt> (disponible en anglais seulement)

RFC 3394:2002, Internet Engineering Task Force (IETF). *Advanced Encryption Standard (AES) Key Wrap Algorithm*. Édité par J. Schaad (Soaring Hawk Consulting) et R. Housley (RSA Laboratories) septembre 2002. Disponible à l'adresse: <http://www.rfc-editor.org/rfc/rfc3394.txt> (disponible en anglais seulement)

RFC 4106:2005, *The Use of Galois/Counter Mode (GCM) in IPsec Encapsulating Security Payload (ESP)* (disponible en anglais seulement)

³ À publier simultanément avec la présente partie de la CEI 62056.

⁴ À publier simultanément avec la présente partie de la CEI 62056.

⁵ À publier simultanément avec la présente partie de la CEI 62056.

NOTE Voir aussi la Bibliographie.

3 Termes, définitions et abréviations

3.1 Termes et définitions

Pour les besoins du présent document, les termes et définitions donnés dans la RFC 4106, les CEI/TR 62051 et CEI/TR 62051-1 s'appliquent, ainsi que les suivants.

3.2 Abréviations

AA	Application Association (Association d'applications)
AAD	Additional Authenticated Data (Données authentifiées supplémentaires) (utilisé avec le chiffrement d'APDU)
AARE	A-Associate Response (Réponse d'association d'applications) – une APDU de l'ACSE
AARQ	A-Associate Request (Demande d'association d'applications) – une APDU de l'ACSE
ACPM	Association Control Protocol Machine (Machine de protocole de commande d'association)
ACSE	Association Control Service Element (Élément de service de contrôle d'association)
AE	Application Entity (Entité d'application)
AES	Advanced Encryption Standard (Norme de chiffrement avancé)
AL	Application Layer (Couche application)
AP	Application Process (Processus d'application)
APDU	Application Layer Protocol Data Unit (Unité de données de protocole d'application)
API	Application Programming Interface (Interface de programmes d'applications)
ASE	Application Service Element (Élément de service d'application)
ASO	Application Service Object (Objet de service d'application)
A-XDR	Adapted Extended Data Representation (Représentation de données étendues adaptée)
base_name	Le short_name correspondant au premier attribut ("logical_name") d'un objet COSEM
BER	Basic Encoding Rules (Règles de codage de base)
CF	Control Function (Fonction de commande)
CL	Connectionless (Sans connexion)
Client	Un poste demandant des services. Dans le cas du profil à 3 couches, orienté connexion et basé sur HDLC, il s'agit du poste maître
.cnf	.confirm service primitive (Primitive de service .confirm)
CO	Connection-oriented (Orienté connexion)
COSEM	Companion Specification for Energy Metering (Spécification d'accompagnement pour le comptage de l'énergie)
COSEM class_id	COSEM interface class identifier (identifiant de classe d'interface COSEM)
Objet COSEM	Instance d'une classe d'interface COSEM
DCS	Data Collection System (Système de collecte de données)

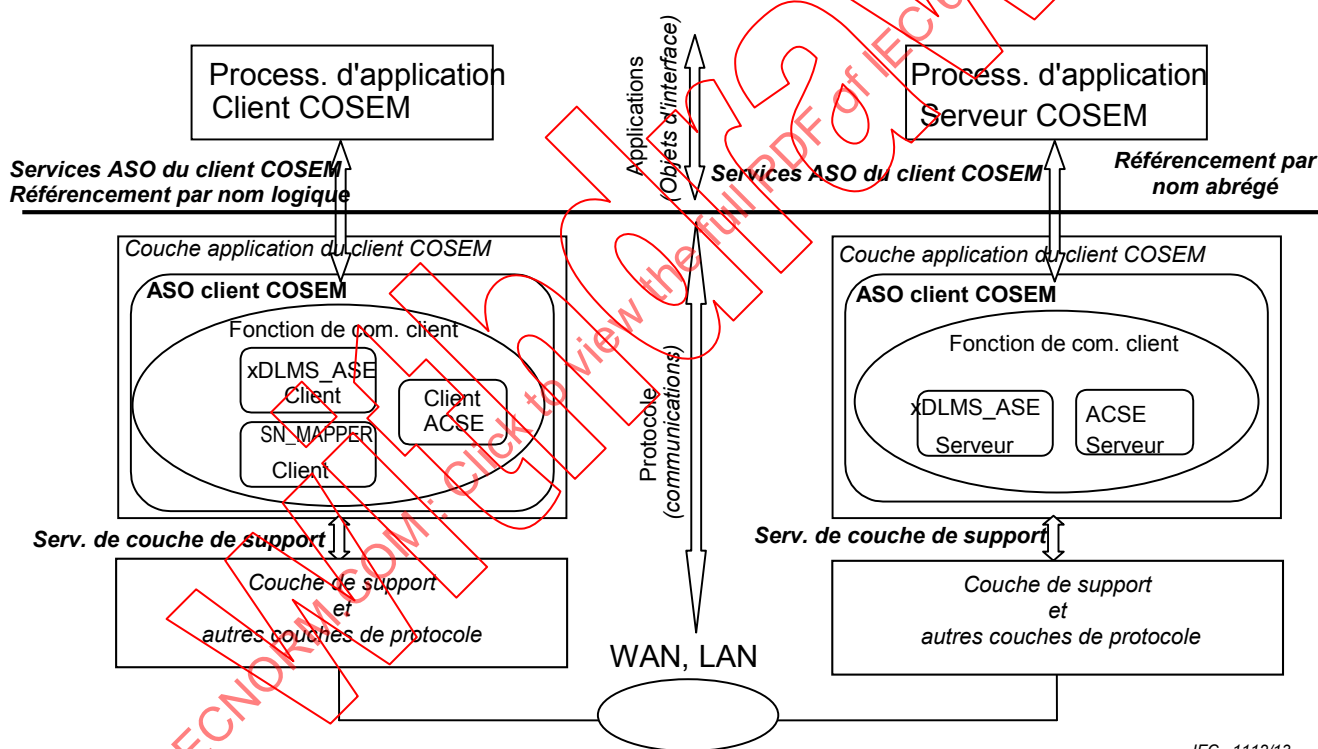
DLMS	Device Language Message Specification (Spécification de message de langage de dispositif)
DLMS UA	DLMS User Association
FIPS	Federal Information Processing Standard (Norme de traitement des informations fédérales)
GCM	Galois/Counter Mode (Mode Galois/Compteur) – un algorithme de chiffrement authentifié avec données associées
GMAC	Spécialisation du GCM pour la génération de codes d'authentification de messages (Message Authentication Code, MAC) sur des données non chiffrées
HDLC	High-level Data Link Control (Commande de liaison de données à haut niveau)
HLS	High Level Security (Sécurité de haut niveau)
HMAC	Keyed-Hash Message Authentication Code (Code d'authentification de message avec hachage par clé)
IEEE	Institute of Electrical and Electronics Engineers (Institut d'ingénieurs en électricité et en électronique)
IETF	Internet Engineering Task Force (Groupe de travail d'ingénierie Internet)
.ind	.indication service primitive (Primitive de service .indication)
IP	Internet Protocol (Protocole Internet)
ISO	International Organization for Standardization (Organisation Internationale de Normalisation)
IV	Initialization Vector (Vecteur d'initialisation)
LDN	Logical Device Name (Nom d'unité logique)
LLC	Logical Link Control (Commande de liaison logique) – sous-couche
LLS	Low Level Security (Sécurité de bas niveau)
LSB	Least Significant Bit (Bit de poids faible)
MAC	Medium Access Control (Contrôle d'accès au support) – sous-couche
MAC	Message Authentication Code (Code d'authentification de message) – cryptographie
maître	Poste central qui prend l'initiative et contrôle le flux de données
MSB	Most Significant Bit (Bit de poids fort)
MSC	Message Sequence Chart (Diagramme de séquence de message)
NIST	National Institute of Standards and Technology (Institut national des normes et de la technologie)
OBIS	Object Identification System (Système d'identification d'objet)
OSI	Open System Interconnection (Interconnexion de systèmes ouverts)
PDU	Protocol Data Unit (Unité de données de protocole)
PHSDU	PH SDU
CPL	Power Line Carrier (Courant porteur sur ligne)
PPP	Point-to-Point Protocol (Protocole point-à-point)
.req	.request service primitive (Primitive de service .request)
.res	.response service primitive (Primitive de service .response)
RLRE	A-Release Response (Réponse de libération A) – une APDU de l'ACSE
RLRQ	A-Release Request (Demande de libération A) – une APDU de l'ACSE
SAP	Service Access Point (Point d'accès au service)

Serveur	Poste fournissant des services. Le dispositif de tarification (compteur) est généralement le serveur, qui fournit les valeurs demandées ou exécute les tâches demandées.
TCP	Transmission Control Protocol (Protocole de contrôle de transmission)
TDEA	Triple Data Encryption Algorithm (Triple algorithme de chiffrement de données)
UDP	User Datagram Protocol (Protocole de datagramme utilisateur)
VAA	Virtual Application Association (Association d'applications virtuelles)
xDLMS_ASE	Extended DLMS Application Service Element (Élément de service d'application DLMS étendu)

4 Vue d'ensemble

4.1 Structure de la couche application COSEM

La structure des couches applications COSEM du client et du serveur est présentée dans la Figure 1.



IEC 1112/13

Figure 1 – Structure des couches application COSEM

Le composant principal de l'AL COSEM est l'objet de service d'application COSEM. Il fournit des services à son usager, le processus d'application COSEM, et utilise les services fournis par la couche inférieure de support. Il contient les trois composants obligatoires du côté client et serveur:

- l'élément de service de contrôle d'association, ACSE;
- l'élément de service d'application DLMS étendu, xDLMS_ASE;
- la fonction de contrôle (CF).

Du côté client, il existe un quatrième élément facultatif, appelé l'ASE SN_MAPPER.

La tâche de l'ACSE consiste à établir, maintenir et libérer les associations d'applications. Pour les besoins des profils de communication DLMS/COSEM orientés connexion (CO), l'ACSE CO indiqué dans les normes ISO/CEI 15953 et ISO/CEI 15954 est utilisé.

La tâche de xDLMS_ASE est de fournir des services de transfert de données entre les AP COSEM. Cet ASE est basé sur la norme DLMS, CEI 61334-4-41. Il a été étendu pour DLMS/COSEM; voir 4.2.3.1. L'objectif principal de DLMS/COSEM est de fournir un modèle d'objet d'interface, orienté domaine de métier, destiné aux dispositifs et aux systèmes de mesure, tout en conservant une compatibilité rétroactive avec la norme DLMS. Pour atteindre ces objectifs, DLMS/COSEM inclut une évolution de DLMS. Tout en restant entièrement conforme avec la norme DLMS, DLMS/COSEM offre un aperçu du compteur plus spécifique au comptage grâce aux objets d'interface COSEM.

Le paragraphe 4.2 de la CEI 62056-6-2:— indique deux méthodes de référencement pour les attributs et les méthodes des objets d'interface COSEM à utiliser sur les serveurs COSEM: le référencement par LN et par SN. Lorsque le référencement par LN est utilisé, le Nom logique des objets COSEM doit être tel que spécifié dans la CEI 62056-6-1. Par conséquent, du côté serveur, deux ensembles de service xDLMS distincts sont indiqués: l'un exclusivement pour le référencement LN et l'autre exclusivement pour le référencement SN. Il est possible de considérer qu'il existe deux xDLMS_ASE distincts: l'un fournissant des services pour le référencement LN et l'autre, pour le référencement SN. L'AL du serveur peut inclure un xDLMS_ASE, l'autre ou les deux.

L'élément CF indique comment les services ASO appellent les primitives de service appropriées de l'ACSE, du xDLMS_ASE et des services de la couche de support.

NOTE Les ASO COSEM du client et du serveur peuvent tous deux contenir d'autres composants de protocole d'application facultatifs.

L'ASE SN_MAPPER facultatif est présent dans l'ASO de l'AL du côté client, lorsque le serveur utilise le référencement SN. Il fournit un mappage entre les services utilisant le référencement LN et le référencement SN. Voir également 6.17.

L'AL COSEM exécute également quelques fonctions de la couche présentation OSI:

- Codage de l'APDU de l'ACSE en BER (ISO/CEI 8825-1). Voir également 7.2.3;
- Codage des APDU xDLMS contenant les services de transfert de données dans A-XDR (CEI 61334-6).

4.2 Services de la couche application COSEM

4.2.1 Services ASO

Les services requis par, ou fournis pour les AP du client et du serveur COSEM au niveau des interfaces logiques avec l'AL COSEM respective, à l'aide de procédures CO, sont répartis en trois catégories:

- établissement et libération d'associations d'applications;
- transfert de données;
- gestion des couches.

4.2.2 Services fournis pour l'établissement et la libération d'associations d'applications

Les services fournis pour l'établissement et la libération des AA sont les suivants:

- COSEM-OPEN;
- COSEM-RELEASE;
- COSEM-ABORT.

Le service COSEM-OPEN permet d'établir des AA. Il est basé sur le service A-ASSOCIATE de l'ACSE. Il entraîne le démarrage de l'utilisation d'une AA par ces procédures ACSE identifiées par la valeur des paramètres `Application_Context_Name`, `Security_Mechanism_Name` et de contexte `xDLMS`. Les AA peuvent être établies de différentes manières:

- Les AA confirmées sont établies par l'intermédiaire d'un échange de messages, à l'aide du service COSEM-OPEN, entre le client et le serveur pour négocier les contextes. Les AA confirmées peuvent être établies entre un client et un serveur uniques;
- Les AA non confirmées sont établies par l'intermédiaire d'un envoi de messages, à l'aide du service COSEM-OPEN, du client vers le serveur, en utilisant les paramètres des contextes potentiellement utilisés par le serveur. Les AA non confirmées peuvent être établies entre un client et un ou plusieurs serveurs;
- Des AA pré-établies peuvent pré-exister. Dans ce cas, le service COSEM-OPEN n'est pas utilisé. Une AA pré-établie peut être confirmée ou non confirmée.

Le service COSEM-RELEASE permet de libérer des AA. Si le service aboutit, il entraîne l'arrêt de l'utilisation de l'AA sans perte d'informations en transit (libération gracieuse). Dans certains profils de communication (par exemple, dans le profil basé sur TCP-UDP/IP), le service COSEM-RELEASE est basé sur le service A-RELEASE de l'ACSE. Dans d'autres profils de communication (par exemple, dans le profil à 3 couches, CO, basé sur HDLC), il existe une relation un-à-un entre une AA confirmée et la connexion à la couche de protocole de support. Par conséquent, les AA peuvent être libérées simplement en déconnectant la connexion à la couche de support correspondante. Les AA pré-établies ne peuvent pas être libérées.

Le service COSEM-ABORT entraîne la libération anormale d'une AA ainsi que la perte potentielle d'informations en transit. Il n'est pas basé sur le service A-ABORT de l'ACSE.

Le service COSEM-OPEN est spécifié en 6.2, le service COSEM-RELEASE en 6.3 et le service COSEM-ABORT en 6.4.

4.2.3 Services fournis pour le transfert de données

4.2.3.1 Élément de service d'application xDLMS

Comme indiqué en 4.1, xDLMS inclut certaines extensions à la norme DLMS, CEI 61334-4-41. Ces extensions définissent des fonctionnalités ajoutées. Les fonctionnalités existantes ne sont pas modifiées. Elles sont élaborées de sorte qu'il n'existe aucun conflit avec la norme DLMS existante et comprennent les éléments suivants:

- des services supplémentaires;
- des types de données supplémentaires;
- un nouveau numéro de version DLMS;
- un nouveau bloc de conformité, et
- l'explication de la signification des paramètres PDU Size.

Les services supplémentaires sont GET, SET, ACTION et EventNotification. Ils permettent de référencer les attributs et les méthodes des objets d'interface COSEM à l'aide du référencement LN. Voir 4.2.3.5 et 4.2.3.6.

Les types de données supplémentaires sont spécifiés dans l'Article 8.

Le nouveau numéro de version DLMS, qui correspond à la première version de l'ASE xDLMS, est 6.

Le nouveau bloc de conformité xDLMS permet les implémentations de serveur DLMS/COSEM optimisées, avec des fonctionnalités étendues. Il se distingue du bloc de conformité DLMS de par sa balise « Application 31 ». Voir 7.3.1 et l'Article 8.

Pour les services utilisant le référencement SN, de nouvelles variantes du paramètre de service Variable_Access_Specification, ainsi que des services Read.response et Write.response ont été ajoutées pour prendre en charge l'accès sélectif et le transfert de bloc.

Pour clarifier la signification des paramètres Max PDU Size utilisables par le client et le serveur, les modifications indiquées dans le Tableau 1 ont été effectuées. Le service xDLMS-Initiate utilise ces noms pour les paramètres PDU Size.

Tableau 1 – Explication de la signification des paramètres PDU Size pour DLMS/COSEM

Avant:	Maintenant:
Page 60, Tableau 3, CEI 61334-4-41:1996	
Proposed Max PDU Size	Client Max Receive PDU Size
Negotiated Max PDU Size	Server Max Receive PDU Size
Page 62, 5^e paragraphe, CEI 61334-4-41:1996	
Le paramètre Proposed Max PDU Size (taille maximale proposée du PDU), de type Unsigned16 (nombre non signé sur 16 bits), propose une longueur maximale exprimée en octets, des PDU de DLMS échangés. La valeur proposée dans une demande Initiate doit être suffisante pour toujours permettre la transmission du PDU "Initiate Error" (erreur d'initialisation).	Le paramètre Client Max Receive PDU Size, de type Unsigned16, contient la longueur maximale, exprimée en octets, pour une PDU DLMS pouvant être envoyée par le serveur. Le client supprimera toute PDU reçue dont la taille est supérieure à cette longueur maximale. La valeur doit être suffisamment élevée pour toujours permettre la transmission des APDU AARE. Les valeurs inférieures à 12 sont réservées. La valeur 0 indique qu'il n'existe aucune limite pour la taille de PDU.
Page 62, dernier paragraphe, CEI 61334-4-41:1996	
Le paramètre Negotiated Max PDU Size (Taille maximale négociée du PDU), de type Unsigned16, contient la longueur maximale exprimée en octets, pour les PDU de DLMS échangés. Une PDU qui serait plus longue que cette longueur maximale ne serait pas considérée. La longueur maximale est traitée comme le minimum de Proposed Max PDU Size et le maximum de la taille de PDU que le VDE-handler peut supporter.	Le paramètre Server Max Receive PDU Size, de type Unsigned16, contient la longueur maximale exprimée en octets pour une PDU DLMS pouvant être envoyée par le client. Le serveur supprimera toute PDU reçue dont la taille est supérieure à cette longueur maximale. Les valeurs inférieures à 12 sont réservées. La valeur 0 indique qu'il n'existe aucune limite pour la taille de PDU.

4.2.3.2 Services de type client/serveur et non-client/serveur

Les services de transfert de données xDLMS sont liés aux attributs et aux méthodes des objets d'interface COSEM, indiqués dans la CEI 62056-6-2, et fournissent l'interface entre le modèle d'application COSEM et les protocoles de communication.

Lorsque le serveur utilise le référencement SN, les attributs et les méthodes des objets d'interface COSEM sont mappés aux variables nommées DLMS. Le mappage est contenu par le ou les objets d'interface Association SN.

Pour accéder aux attributs et aux méthodes, les services de type client/serveur sont utilisés: le client demande des services et le serveur les fournit.

Il existe également un service de type non-client/serveur non sollicité. Ce service est demandé par le serveur, lorsqu'un événement se produit, afin d'informer le client de la valeur d'un ou de plusieurs attributs, comme s'ils avaient été demandés par le client. Il s'agit d'un service non confirmé.

4.2.3.3 Méthodes de référencement et mappage de services

Comme spécifié en 4.1, deux ensembles de services distincts sont disponibles: l'un pour le référencement LN et l'autre pour le référencement SN.

Du côté client, afin de traiter les différentes méthodes de référencement de manière transparente pour l'AP, l'AL ne fournit qu'un seul ensemble de services utilisant le référencement LN. L'utilisation d'un ensemble de services unique et standardisé entre les AP du client COSEM et le protocole de communication, en masquant les caractéristiques des serveurs COSEM utilisant différentes méthodes de référencement, permet d'indiquer une interface de programmes d'applications (API). Il s'agit d'une interface explicitement spécifiée qui correspond à cet ensemble de services pour des applications s'exécutant dans un environnement informatique donné (par exemple, Windows, UNIX, etc.). À l'aide de cette spécification (publique) d'API, des applications client peuvent être développées sans connaître les caractéristiques d'un serveur donné.

Du côté serveur, des services utilisant le référencement LN et/ou le référencement SN peuvent être fournis.

Dans le cas des AA confirmées, la méthode de référencement et l'ensemble de services à utiliser sont négociés durant la phase d'établissement de l'AA via le contexte d'application COSEM. Voir 7.2.2.2 et le bloc de conformité. Voir 7.3.1. Elle ne doit pas changer pendant la durée de vie de l'AA établie. L'utilisation de services LN ou SN avec une AA donnée est exclusive.

Dans le cas d'AA non confirmées et pré-établies, l'AL du client est censée connaître la méthode de référencement et l'ensemble de services pris en charge par le serveur.

Lorsque le serveur utilise le référencement LN, les services sont identiques des deux côtés. Lorsque le serveur n'utilise pas le référencement LN, un mappage entre les services côté client et côté serveur a lieu. Comme expliqué en 4.1, si le serveur utilise le référencement SN, ce mappage est effectué par l'ASE SN_MAPPER du client. Voir également 6.17.

4.2.3.4 Services confirmés et non confirmés

Dans les AA confirmées, les services de transfert de données de type client/serveur peuvent être appelés de manière confirmée ou non confirmée. Dans les AA non confirmées, les services de transfert de données de type client/serveur peuvent être appelés de manière non confirmée uniquement. Voir également 7.3.2.

4.2.3.5 COSEM, services de type client/serveur

Les services de transfert de données de type client/serveur COSEM pour le référencement LN sont les suivants:

- le service GET, utilisé pour lire la valeur d'un ou de plusieurs attributs des objets d'interface COSEM. Voir 6.6;
- le service SET, utilisé pour écrire la valeur d'un ou de plusieurs attributs des objets d'interface COSEM. Voir 6.7;
- le service ACTION, utilisé pour appeler une ou plusieurs méthodes des objets d'interface COSEM. L'appel de méthodes peut impliquer l'envoi de paramètres de service et le renvoi de données. Voir 6.8.

Les services de transfert de données de type client/serveur COSEM pour le référencement SN sont les suivants:

- le service Read, utilisé pour lire la valeur d'un ou de plusieurs attributs, ou pour appeler une ou plusieurs méthodes des objets d'interface COSEM lorsqu'un retour de paramètres est attendu. Il s'agit d'un service confirmé. Voir 6.12;

- le service Write, utilisé pour écrire la valeur d'un ou de plusieurs attributs, ou pour appeler une ou plusieurs méthodes des objets d'interface COSEM lorsqu'aucun retour de paramètres n'est attendu. Il s'agit d'un service confirmé. Voir 6.13;
- le service UnconfirmedWrite, utilisé pour écrire la valeur d'un ou de plusieurs attributs, ou pour appeler une ou plusieurs méthodes des objets d'interface COSEM. Il s'agit d'un service non confirmé. Voir le paragraphe 6.14.

4.2.3.6 Services de notification d'événements

Pour prendre en charge la notification d'événements, COSEM fournit également des services non sollicités, de type non-client/serveur:

- avec le référencement LN, le service EventNotification. Voir 6.9;
- avec le référencement SN, le service InformationReport. Voir 6.15.

4.2.3.7 Identification des appels de services: paramètre Invoke_Id

Dans le modèle client/serveur, les demandes sont envoyées par le client et les réponses, par le serveur. Le client est autorisé à envoyer plusieurs demandes avant de recevoir la réponse aux précédentes. Par conséquent, pour pouvoir identifier quelle réponse correspond à quelle demande, il est nécessaire d'inclure une référence dans la demande.

Le paramètre Invoke_Id est utilisé à cet effet. La valeur de ce paramètre est attribuée par le client de sorte que chaque demande contient un Invoke_Id différent. Le serveur doit copier le paramètre Invoke_Id dans la réponse correspondante.

En tant que service de type non-client/serveur, le service EventNotification ne contient pas le paramètre Invoke_Id.

Cette fonctionnalité est disponible uniquement avec le référencement LN.

4.2.3.8 Priorité des appels de services: paramètre Priority

Pour les services de transfert de données utilisant le référencement LN, deux niveaux de priorité sont disponibles: normal (FALSE) et élevé (TRUE). Cette fonctionnalité permet de recevoir une réponse à nouvelle demande avant que la réponse à une demande précédente ne soit achevée.

En règle générale, le serveur traite les demandes de service entrantes dans l'ordre de réception (FIFO, First In-First Served ou Premier entré-Premier traité). Cependant, une demande pour laquelle le paramètre Priority est défini sur ÉLEVÉ doit être traitée avant les demandes précédentes définies sur NORMAL. L'indicateur de priorité de la réponse doit être identique à celui de la demande correspondante. La gestion de la priorité est une fonctionnalité négociable. Voir 7.3.1.

NOTE 1 Étant donné que les appels de services sont identifiés par un Invoke_Id, les services dont la priorité est identique peuvent être traités dans n'importe quel ordre.

NOTE 2 Si la fonctionnalité n'est pas prise en charge, les demandes dont la priorité est ÉLEVÉE sont traitées avec une priorité NORMALE.

Cette fonctionnalité n'est pas disponible avec les services utilisant le référencement SN. Le serveur traite les services sur une base FIFO.

4.2.3.9 Accès sélectif

Dans le cas de certaines classes d'interface COSEM, un accès sélectif aux attributs est disponible, ce qui signifie qu'il est possible d'accéder soit à tout l'attribut, soit à une partie sélectionnée de celui-ci. À cet effet, les sélecteurs et les paramètres d'accès sont indiqués dans le cadre de la spécification des attributs concernés.

Pour utiliser cette option, les services liés aux attributs peuvent être appelés avec ces paramètres de sélection d'accès. Dans le cas du référencement LN, cette fonctionnalité est appelée Accès sélectif. Voir 6.6 et 6.7. Il s'agit d'une fonctionnalité négociable. Voir 7.3.1. Dans le cas du référencement SN, cette fonctionnalité est appelée Accès paramétré. Voir 6.12, 6.13 et 6.14. Il s'agit d'une fonctionnalité négociable. Voir 7.3.1.

4.2.3.10 Références multiples

Dans un appel de services de transfert de données, il est possible de faire référence à une ou plusieurs méthodes ou des attributs. L'utilisation de références multiples est une fonctionnalité négociable. Voir 7.3.1.

4.2.3.11 Référencement Attribute_0

Avec les services GET et SET, une fonctionnalité spéciale appelée référencement Attribute_0 est disponible. Par convention, les attributs des objets d'interface COSEM sont numérotés de 1 à n, où Attribute_1 est le nom logique de l'objet d'interface COSEM. Attribute_0 a une signification spéciale: il référence tous les attributs possédant un index positif (attributs publics). L'utilisation du référencement Attribute_0 avec le service GET est expliquée en 6.6 et avec le service SET, en 6.7. Le référencement Attribute_0 est une fonctionnalité négociable. Voir 7.3.1.

NOTE Comme indiqué dans la CEI 62056-6-2:—, 4.1 et 4.2, les fabricants peuvent ajouter des méthodes et/ou des attributs propriétaires à n'importe quel objet, à l'aide de numéros négatifs.

4.2.3.12 Transfert de paramètres de service longs

Les services de transfert de données sont contenus dans les APDU échangées entre les AL homologues, sous forme codée. Dans certains cas, les données peuvent être plus longues que le paramètre Client/Server Max Receive PDU Size négocié. Pour transférer des données de type « long », deux mécanismes de transport sont disponibles:

- a) le transfert de données de type long à l'aide du protocole de transfert de blocs de l'AL. Ce mécanisme peut être utilisé avec n'importe quel service de type client/serveur confirmé. Il s'agit d'une fonctionnalité négociable. Voir 7.3.1;
- b) le transfert de données de type long de manière transparente vers l'AL. Cette fonctionnalité peut être utilisée lorsque la ou les couches de support offrent une possibilité de segmentation. Voir l'Annexe A.

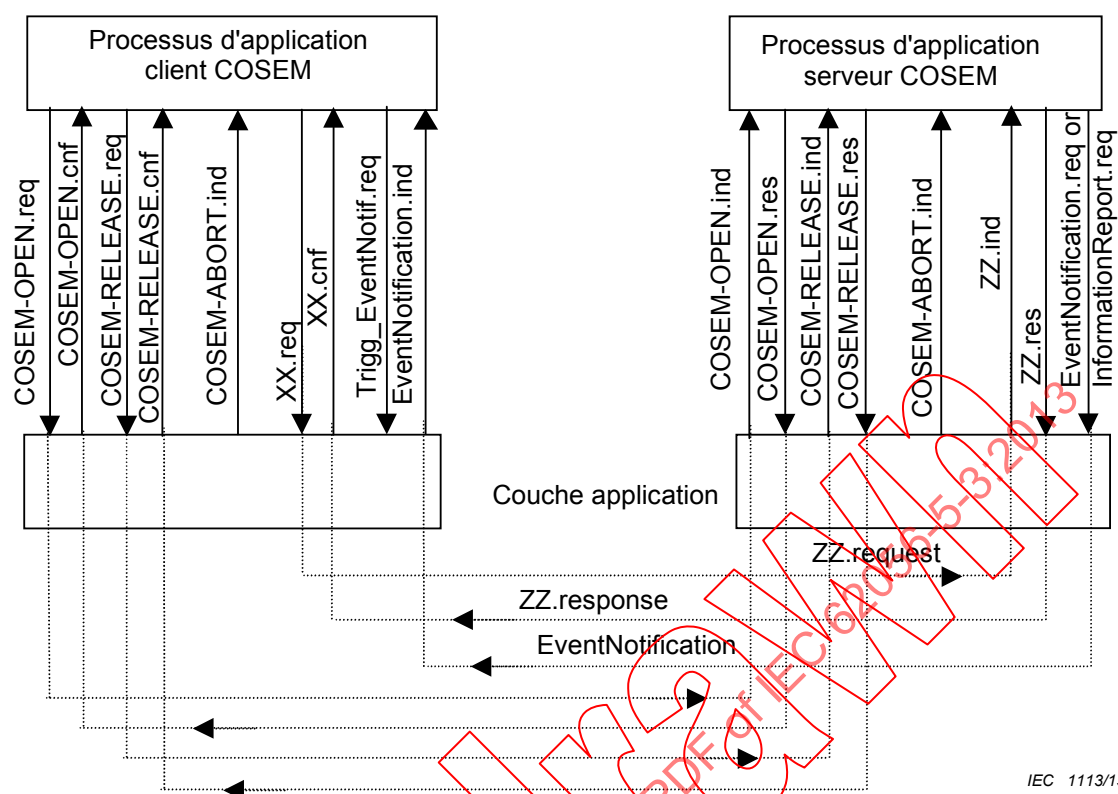
NOTE Il n'existe aucun mécanisme dans le transfert de blocs de l'AL permettant de renvoyer ou de redemander un bloc manquant ou corrompu. Un tel mécanisme peut être une fonctionnalité de transfert de données de type long, gérée de façon transparente par les couches de support.

4.2.4 Services de gestion de couche

Les services de gestion de couche ont une importance locale uniquement. Par conséquent, l'indication de ces services n'entre pas dans le domaine d'application de la présente spécification auxiliaire. Le service SetMapperTable spécifique est défini en 6.16.

4.2.5 Récapitulatif des services de la couche application COSEM

Un récapitulatif des services disponibles en haut de l'AL COSEM est présenté dans la Figure 2. Les services de gestion de couche ne sont pas présentés. Même si les primitives de service sont différentes du côté client et serveur, les APDU sont identiques. La syntaxe abstraite de l'ACSE et des APDU COSEM est indiquée à l'Article 8.



NOTE XX et ZZ désignent les services de transfert de données de type client/serveur. Ces services sont différents du côté client et du côté serveur si le serveur n'utilise pas le référencement LN. Voir 6.17.

Figure 2 – Récapitulatif des services de l'AL COSEM

4.3 Protocoles de la couche application COSEM

Les protocoles de l'AL COSEM indiquent les procédures utilisées pour le transfert d'informations, pour le contrôle et l'authentification de l'AA à l'aide de procédures ACSE orientées connexion et pour le transfert de données entre les clients et les serveurs COSEM à l'aide de procédures xDLMS. Par conséquent, le protocole de l'AL est basé sur les protocoles ACSE et DLMS, comme indiqué dans l'ISO/CEI 15954 et l'CEI 61334-4-41 (y compris les extensions pour DLMS/COSEM) respectivement. Les procédures sont définies en termes:

- d'interactions entre les machines de protocole ACSE et xDLMS homologues par l'intermédiaire de l'utilisation des services de la couche de protocole de support;
- d'interactions entre les machines de protocole ACSE et xDLMS et leur usager;
- de syntaxe abstraite des APDU utilisant ASN.1, comme indiqué dans l'ISO/CEI 8824-1.

Les protocoles de l'AL COSEM sont spécifiés à l'Article 7.

5 Sécurité des informations dans DLMS/COSEM

NOTE L'Article 5 est basé sur des parties de documents du NIST. Réimprimé avec l'aimable autorisation du NIST (National Institute of Standards and Technology), Technology Administration, U.S. Department of Commerce. Non protégeable aux États-Unis.

5.1 Définitions

5.1.1

confidentialité

garantir les restrictions autorisées sur l'accès aux informations et leur divulgation, y compris les moyens de protection des informations privées et exclusives. Une perte de la confidentialité correspond à la divulgation non autorisée d'informations

[SOURCE: FIPS PUB 199]

5.1.2

intégrité

protéger contre une modification abusive ou une destruction d'information; inclut la garantie de la non-répudiation et de l'authenticité des informations. Une perte d'intégrité correspond à la modification ou la destruction non autorisée d'informations

[SOURCE: FIPS PUB 199]

5.2 Généralités

La confidentialité et l'intégrité font partie des exigences clés lorsque des systèmes ouverts se connectent à des supports publics. Avec l'augmentation de la puissance informatique, la nécessité d'avoir des méthodes cryptographiques sûres est d'autant plus importante. DLMS/COSEM fournit deux fonctionnalités principales de sécurité des informations pour accéder aux données et les transporter:

- *la sécurité de l'accès aux données* contrôle l'accès aux données stockées sur un serveur DLMS/COSEM. Voir 5.3;
- *La sécurité de transport des données* permet à l'expéditeur d'appliquer la protection cryptographique aux APDU xDLMS envoyées pour garantir la confidentialité et l'intégrité. Elle requiert des APDU chiffrées. Le destinataire peut supprimer ou vérifier cette protection. Voir 5.4.

Ces fonctionnalités de sécurité des informations sont en partie fournies par l'AL COSEM et par des objets COSEM:

- lors de l'établissement d'une AA, deux contextes sont négociés à l'aide des services ACSE: le *contexte d'application* et le *contexte d'authentification*. Le contexte d'application détermine si des APDU chiffrées peuvent être utilisées ou non. Le contexte d'authentification détermine le niveau de sécurité de l'accès aux données. Les APDU de l'ACSE ne sont pas protégées de manière cryptographique;

Le champ user-information des APDU de l'ACSE doit être protégé de manière cryptographique, tel que déterminé par l'application-context et la politique de sécurité.

- une fois une AA établie avec succès, les services COSEM peuvent être utilisés pour accéder aux attributs et aux méthodes des objets COSEM, en fonction des droits d'accès valides dans l'association donnée. Ces services sont fournis par les APDU xDLMS et celles-ci peuvent être protégées de manière cryptographique, c'est-à-dire authentifiées et/ou cryptées, en fonction de la politique de sécurité en vigueur. La protection cryptographique est appliquée, supprimée ou vérifiée par l'AL COSEM. L'AP COSEM de l'expéditeur informe, via des paramètres de service, l'AL COSEM de la protection à appliquer à l'APDU à envoyer. L'AL COSEM du destinataire informe l'AP COSEM de la protection qui a été appliquée à l'APDU reçue.

Les méthodes de sécurité des messages décrites dans la présente partie de la CEI 62056 ont été sélectionnées parmi les normes indiquées par le NIST et l'IETF.

Pour obtenir un aperçu de la cryptographie, voir l'Annexe E.

5.3 Sécurité de l'accès aux données

5.3.1 Vue d'ensemble

La sécurité de l'accès aux données concerne l'accès aux données basé sur les rôles dans un dispositif DLMS/COSEM.

Elle est gérée par les objets Association LN/Association SN. Chaque serveur COSEM, c'est-à-dire un dispositif logique, peut avoir à gérer des AA avec différents clients, chacun d'entre eux ayant un rôle différent et donc des droits d'accès différents. Chaque AA est identifiée par une paire d'adresses de couches inférieures. Chaque objet Association fournit une liste d'objets visibles dans cette AA spécifique et des droits d'accès à ses attributs et méthodes.

Pour pouvoir accéder aux données, le client doit être correctement authentifié. Lors de l'établissement de l'AA, un contexte d'authentification est négocié entre le client et le serveur. Celui-ci indique l'authentification requise des homologues, et, le cas échéant, l'algorithme de sécurité permettant de vérifier l'authentification. Trois niveaux de sécurité de l'accès aux données sont fournis:

- le niveau de sécurité le plus faible (pas de sécurité);
- le niveau de sécurité faible (LLS), et
- le niveau de sécurité élevé (HLS).

5.3.2 Niveau de sécurité le plus faible (pas de sécurité)

L'objectif du niveau de sécurité le plus faible est de permettre au client de récupérer quelques informations élémentaires. Ce contexte d'authentification ne requiert pas d'authentification des homologues. Il permet l'accès direct aux données situées sur le serveur, en fonction des droits d'accès disponibles dans l'AA donnée.

5.3.3 Niveau de sécurité faible (LLS)

L'objectif du niveau de sécurité faible est de permettre l'authentification des clients en vérifiant le mot de passe fourni. Le serveur n'est pas authentifié. Ce contexte d'authentification est généralement utilisé lorsque le canal de communication offre une sécurité adéquate, qui permet d'éviter les interceptions non autorisées et la reproduction (de mots de passe) de messages.

Le client doit fournir le mot de passe correct durant le processus d'établissement de l'AA. Si le mot de passe est correct, l'AA est établie et le client peut accéder aux données, en fonction des droits d'accès disponibles dans l'AA donnée. Sinon, l'AA n'est pas établie.

Le processus d'authentification LLS est pris en charge par le service COSEM-OPEN de l'AL COSEM. Voir 6.2. Le processus fonctionne de la manière suivante:

- le client transmet un "secret" (par exemple, un mot de passe) au serveur à l'aide du paramètre "Calling_Authentication_Value" de la primitive de service COSEM-OPEN.request;
- le serveur vérifie si le "secret" reçu correspond à l'identification du client;
- si oui, le client est authentifié et l'AA peut être établie. Si non, l'AA est rejetée;
- le résultat et les informations de diagnostic sont renvoyés par le serveur à l'aide de la primitive de service COSEM-OPEN.response.

Les objets Association offrent une méthode/un attribut pour modifier le "secret". Voir la CEI 62056-6-2:—, 5.3.1 et 5.3.2.

Le processus d'authentification avec LLS est présenté dans la partie gauche de la Figure 3.

5.3.4 Niveau de Sécurité élevé (HLS)

L'objectif du niveau de sécurité élevé est de permettre l'authentification mutuelle du client et du serveur participant à une association. Ce contexte d'authentification est généralement utilisé lorsque le canal de communication n'offre aucune sécurité intrinsèque et des précautions doivent être prises contre les interceptions non autorisées et la reproduction (de mots de passe) de messages.

HLS requiert que le client et le serveur s'authentifient mutuellement l'un auprès de l'autre. Ce processus s'effectue en quatre passes, qui impliquent l'échange de demandes d'accès durant l'établissement de l'AA, puis l'échange des résultats du traitement de ces demandes d'accès à l'aide de méthodes cryptographiques. Si l'authentification a lieu, le client peut accéder aux données, en fonction des droits d'accès disponibles dans l'AA donnée, et accepte les données provenant du serveur. Sinon, l'AA n'est pas établie.

Le processus d'authentification HLS est pris en charge par le service COSEM-OPEN (voir 6.2) et par les objets Association LN/SN comme suit:

Passe 1: Le client transmet un "défi" CtoS (par exemple, une chaîne aléatoire) au serveur;

Passe 2: Le serveur transmet un "défi" StoC (par exemple, une chaîne aléatoire) au client;

La longueur des défis doit être comprise entre 8 octets et 64 octets.

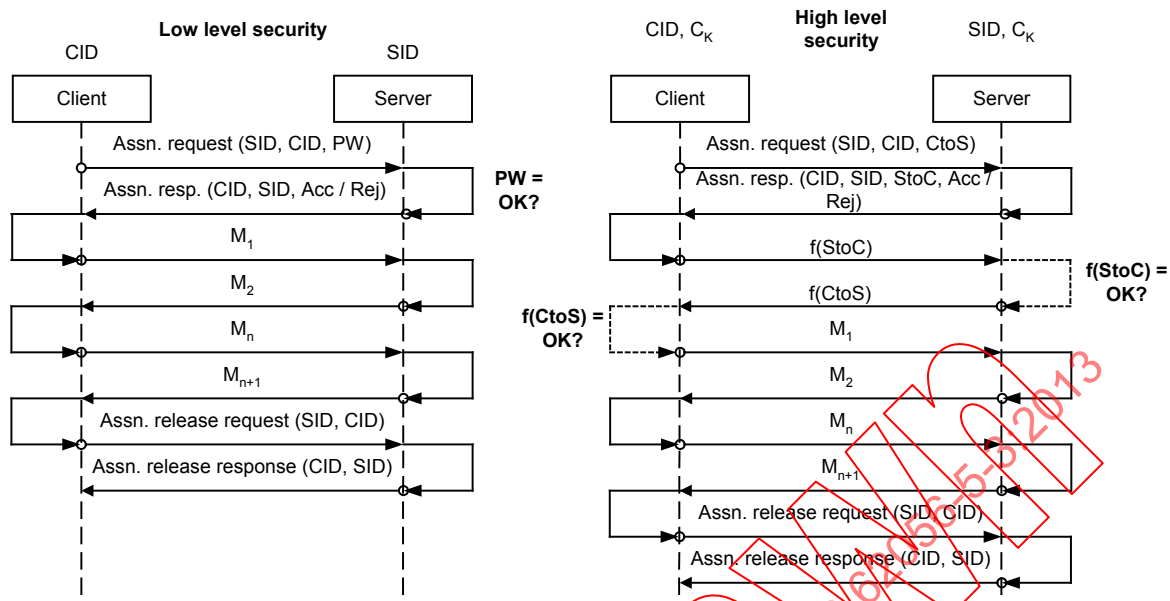
Passe 3: Le client traite le défi StoC en fonction des règles du mécanisme d'authentification HLS négocié:

- dans le cas du mécanisme authentication_mechanism_id(2) HLS, la méthode de traitement du défi est un secret (par exemple, le chiffrement à l'aide d'une clé secrète). Il s'agit du secret HLS qui est connu du client et du serveur;
- dans le cas du mécanisme authentication_mechanism_id(3) HLS, le client concatène vers la droite la StoC reçue durant la passe 2 ainsi que le secret HLS, connu du client et du serveur, puis génère une empreinte à l'aide de l'algorithme MD5. comme spécifié dans RFC 1321;
- dans le cas du mécanisme authentication_mechanism_id(4) HLS, le processus est identique, mais le client génère une empreinte à l'aide de l'algorithme SHA-1. La FIPS PUB 180-1 s'applique à cet égard;
- dans le cas du mécanisme authentication_mechanism_id(5) HLS, une étiquette d'authentification est calculée à l'aide de GMAC. Voir 5.4.8.4.

Le résultat (f(StoC)) est renvoyé au serveur. Le serveur vérifie si f(StoC) est le résultat du traitement correct et, le cas échéant, accepte l'authentification du client.

Passe 4: Si le client est authentifié, le serveur traite le défi CtoS de la même manière que décrit dans la passe 3. Le résultat (f(CtoS)) est renvoyé au client. Le client vérifie si f(CtoS) est le résultat du traitement correct et, le cas échéant, accepte l'authentification du serveur.

Le processus d'authentification avec HLS est présenté dans la partie droite de la Figure 3.



CID: Client address, SID: Server address, PW: Password, C_K : shared secret,
CtoS: client challenge to server, StoC: server challenge to client

IEC 1114/13

Légende

Anglais	Français
Low level security	Niveau de sécurité faible
CID	CID
SID	SID
Client	Client
Server	Serveur
Assn. request (SID, CID, PW)	Demande Assn. (SID, CID, MDP)
Assn. resp. (CID, SID, Acc/Rej)	Réponse Assn. (CID, SID, Acc./Rej.)
PW = OK?	MDP = OK?
M_1	M_1
M_2	M_2
M_3	M_3
Assn. release request (SID, CID)	Demande de libération Assn. (SID, CID)
Assn. release response (SID, CID)	Réponse de libération Assn. (SID, CID)
High level security	Niveau de sécurité élevé
CID, C_K	CID, C_K
Assn. request (SID, CID, CtoS)	Demande Assn. (SID, CID, CtoS)
Assn. resp. (CID, SID, StoC, Acc/Rej)	Réponse Assn. (CID, SID, StoC, Acc./Rej.)
$f(StoC)$	$f(StoC)$
$f(StoC) = OK?$	$f(StoC) = OK?$
$f(CtoS)$	$f(CtoS)$
$f(CtoS) = OK?$	$f(CtoS) = OK?$
M_1	M_1
M_2	M_2

Anglais	Français
M ₃	M ₃
Assn. release request (SID, CID)	Demande de libération Assn. (SID, CID)
Assn. release response (SID, CID)	Réponse de libération (SID, CID)
CID = Client address	CID = Adresse du client
SID = Server address	SID = Adresse du serveur
PW = Password	MDP = Mot de passe
C _K = Shared secret	C _K = Secret partagé
CtoS = Client challenge to Server	CtoS = Demande d'accès du client vers le serveur
StoC = Server challenge to client	StoC = Demande d'accès du serveur vers le client

Figure 3 – Authentification LLS et HLS

La passe 1 est prise en charge par la primitive de service COSEM-OPEN.request de l'AL du client. Le paramètre "Security_Mechanism_Name" contient l'identifiant du mécanisme HLS et le paramètre "Calling_Authentication_Value", la demande d'accès CtoS.

La passe 2 est prise en charge par la primitive de service COSEM-OPEN.response de l'AL du serveur. Le paramètre "Security_Mechanism_Name" contient l'identifiant du mécanisme HLS et le paramètre "Responding_Authentication_Value", la demande d'accès StoC.

Après la passe 2, l'AA est formellement établie, mais l'accès du client est restreint à la méthode "reply_to_HLS_authentication" de l'objet "Association" actuel.

La passe 3 et la passe 4 sont prises en charge par la méthode reply_to_HLS_authentication de l'objet ou des objets Association. Voir CEI 62056-6-2:—, 5.3.1 et 5.3.2. Si les deux passes sont exécutées avec succès, l'accès complet est accordé en fonction de l'AA actuelle. Sinon, soit le client, soit le serveur annule l'AA.

En outre, l'objet Association SN/LN offre la méthode pour modifier le "secret" HLS: change_HLS_secret.

REMARQUE Une fois que le client a émis la méthode change_HLS_secret () ou change_LLS_secret (), il s'attend à une réponse du serveur confirmant la modification du secret. Il est possible que le serveur transmette la confirmation, mais en raison de problèmes de communication, la confirmation n'est pas reçue du côté client. Par conséquent, le client ne sait pas si le secret a été modifié ou non. Pour des raisons de simplicité, le serveur n'offre pas de prise en charge spéciale pour ce cas. En d'autres termes, c'est au client de résoudre ce problème.

5.4 Sécurité de transport des données

5.4.1 Application, suppression ou vérification de la protection: chiffrement et déchiffrement

Pour la sécurité de transport des données, la protection cryptographique peut être appliquée avant d'envoyer une APDU xDLMS, comme déterminé par la politique de sécurité (voir 5.4.3), puis supprimée ou vérifiée après la réception d'une APDU.

NOTE La protection cryptographique des données stockées n'entre pas dans le domaine d'application de la présente spécification auxiliaire.

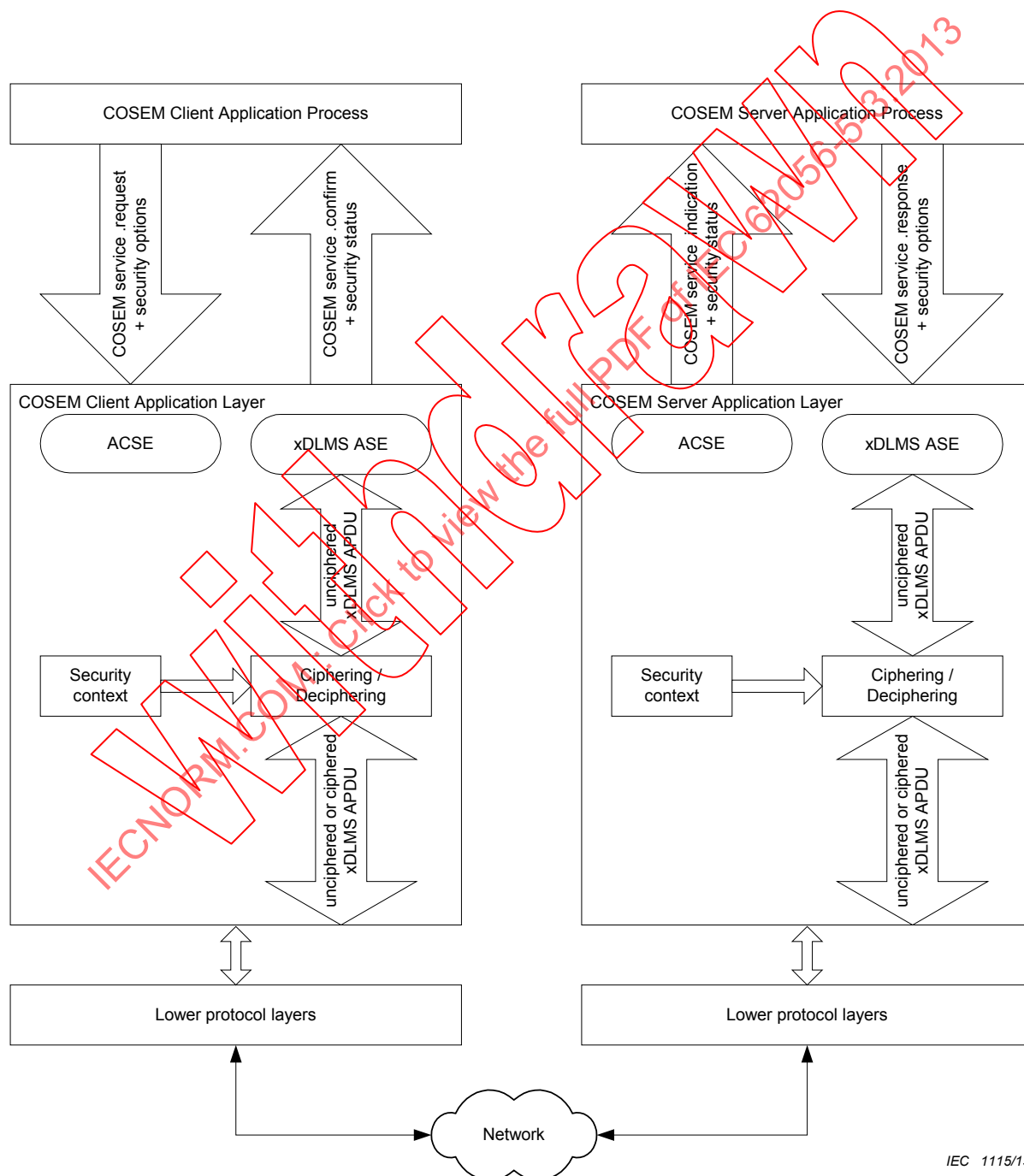
L'application de la protection est effectuée par l'intermédiaire du chiffrement. Le déchiffrement supprime ou vérifie la protection, tout en restaurant l'APDU xDLMS d'origine. Le chiffrement et le déchiffrement sont effectués par l'AL COSEM, comme présenté dans la Figure 4.

Lorsqu'une demande de service ou une primitive de réponse COSEM est appelée par l'AP du client ou du serveur respectivement, les paramètres de service incluent le paramètre

Security_Options. Ce paramètre informe l'AL de la sécurité requise à appliquer à l'APDU xDLMS qui contient la primitive de service et peut identifier les éléments du matériel de sécurité à utiliser.

De la même manière, une indication de service ou une primitive de confirmation COSEM inclut un paramètre **Security_Status**. Ce paramètre informe l'AP de la sécurité appliquée à l'APDU xDLMS qui contient la primitive de service et peut inclure des informations sur les éléments du matériel de sécurité utilisé. L'étiquette d'authentification, le cas échéant, est également transmise.

Pour les spécifications de service COSEM, voir l'Article 6.



Légende

Anglais	Français
COSEM Client Application Process	Processus d'application client COSEM
COSEM service .request + security options	Service COSEM .request + options de sécurité
COSEM service .confirm + security status	Service COSEM .confirm + statut de sécurité
COSEM Client Application Layer	Couche application client COSEM
ACSE	ACSE
xDLMS ASE	ASE xDLMS
Unciphered xDLMS APDU	APDU xDLMS non chiffrée
Security context	Contexte de sécurité
Ciphering/Deciphering	Chiffrement/Déchiffrement
Unciphered or ciphered xDLMS APDU	APDU xDLMS non chiffrée ou chiffrée
Lower protocol layers	Couches de protocole inférieures
Network	Réseau
COSEM Server Application Process	Processus d'application serveur COSEM
COSEM service .indication + security status	Service COSEM .indication + statut de sécurité
COSEM service .response + security options	Service COSEM .response + options de sécurité
COSEM Server Application Layer	Couche application serveur COSEM
ACSE	ACSE
xDLMS ASE	ASE xDLMS
Unciphered xDLMS APDU	APDU xDLMS non chiffrée
Security context	Contexte de sécurité
Ciphering/Deciphering	Chiffrement/Déchiffrement
Unciphered or ciphered xDLMS APDU	APDU xDLMS non chiffrée ou chiffrée
Lower protocol layers	Couches de protocole inférieures

Figure 4 – Sécurité de transport des données dans DLMS/COSEM**5.4.2 Contexte de sécurité**

Le contexte de sécurité définit les attributs de sécurité pertinents pour le processus de chiffrement/déchiffrement:

- la politique de sécurité en vigueur, qui détermine le type de protection à appliquer. Voir 5.4.3;
- la suite de sécurité, qui indique l'algorithme ou les algorithmes de sécurité. Voir 5.4.4;
- le matériel de sécurité concerné pour la suite de sécurité donnée, y compris les éléments tels que les clés de chiffrement de bloc, les clés d'authentification, les vecteurs d'initialisation, etc. Voir 5.4.5.

5.4.3 Politique de sécurité

Les politiques de sécurité suivantes sont indiquées:

- la sécurité n'est pas imposée;
- tous les messages sont authentifiés;
- tous les messages sont cryptés;
- tous les messages sont authentifiés et cryptés.

La politique de sécurité est conservée par l'attribut security_policy de l'objet de configuration Security. Voir la CEI 62056-6-2:—, 5.3.5.

L'accès authentifié peut être requis de manière sélective: il est possible d'accéder à certains attributs et méthodes uniquement à l'aide de messages authentifiés. Celui-ci est indiqué par le mode d'accès à chaque attribut et méthode. Voir la CEI 62056-6-2:—, 5.3.1 et 5.3.2 respectivement. Par conséquent, les APDU xDLMS authentifiées peuvent être utilisées, dans un contexte d'application chiffré, même lorsque la politique de sécurité en vigueur ne requiert pas que tous les messages soient authentifiés.

Notez que si l'accès authentifié est requis, mais que le contexte d'application n'est pas crypté, les attributs et les méthodes nécessitant une authentification ne sont pas accessibles. Si le client tente d'envoyer une APDU chiffrée dans un contexte d'application non crypté, la couche application du serveur doit la rejeter. Dans le cas du référencement SN, la réponse doit être l'APDU ConfirmedServiceError, qui contient l'étiquette de l'APDU rejetée. Dans le cas du référencement LN, la réponse doit être une APDU ExceptionResponse.

D'un autre côté, le chiffrement s'applique en règle générale à tous les messages d'une AA donnée: si la politique de sécurité requiert que tous les messages soient cryptés, toutes les APDU doivent être chiffrées. En outre, elles peuvent être authentifiées.

Lorsque la politique de sécurité requiert que tous les messages soient authentifiés et cryptés, seules les APDU chiffrées et authentifiées peuvent être utilisées.

Les messages protégés par une sécurité plus élevée que celle requise par la politique de sécurité sont toujours autorisés (à condition que le contexte d'application négocié les autorise).

5.4.4 Suite de sécurité

Une suite de sécurité détermine l'algorithme ou les algorithmes cryptographiques utilisés pour la sécurité des messages. Une suite de sécurité est identifiée par un identifiant de la suite de sécurité (Security Suite ID). Voir le Tableau 2. Actuellement, une suite de sécurité est spécifiée, le mode GCM avec AES-128. Dans cette suite de sécurité, les clés globales (voir 5.4.7.3.3) sont protégées lors du transport à l'aide de l'algorithme de chiffrement de clés AES-128.

Tableau 2 – Suites de sécurité

Identifiant de la suite de sécurité	Algorithme d'authentification	Algorithme de chiffrement	Méthode de transport de clé
0	AES-GCM-128	AES-GCM-128	Encapsulation de clé à l'aide de l'algorithme de chiffrement AES-128
Tous les autres réservés	–	–	–

5.4.5 Matériel de sécurité

Les éléments du matériel de sécurité sont les suivants:

- une clé de chiffrement de bloc, symbolisée par *EK*;
- une clé d'authentification, symbolisée par *AK*;
- un vecteur d'initialisation, symbolisé par *IV*.

Les éléments dépendent de la suite de sécurité. Pour AES-GCM-128, ils sont définis en 5.4.8.

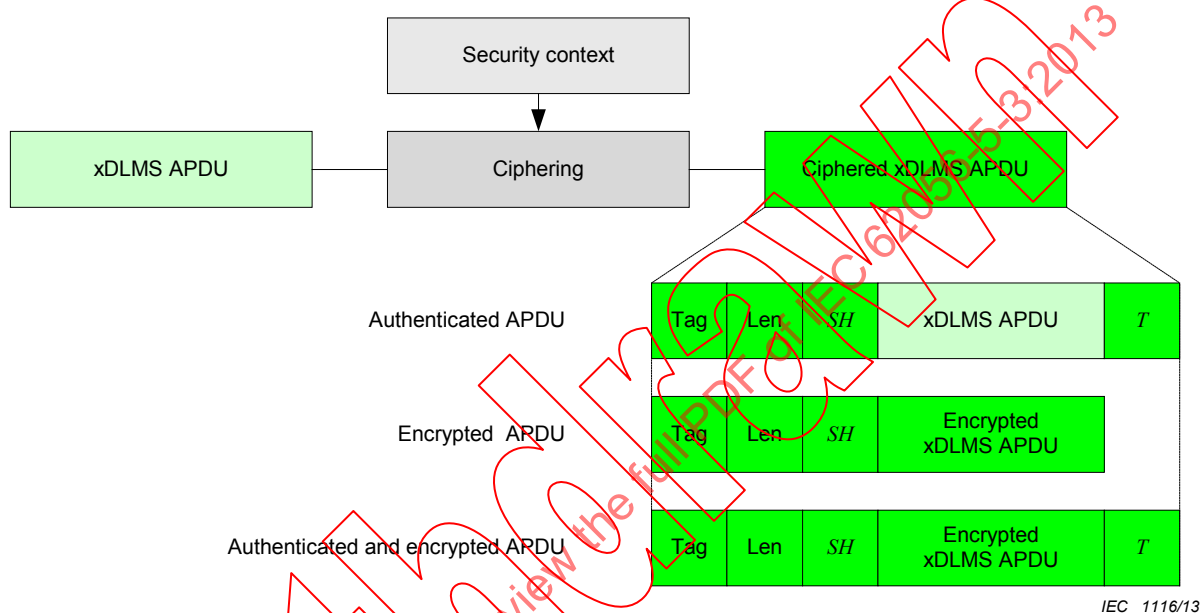
5.4.6 APDU xDLMS chiffrées

Les APDU xDLMS chiffrées peuvent être utilisées dans un contexte d'application COSEM chiffré uniquement. Une APDU chiffrée est toujours une chaîne d'octets.

Dans un contexte d'application COSEM chiffré, les APDU non chiffrées peuvent également être utilisées.

La structure de l'APDU chiffrée dépend du type de chiffrement appliqué. Voir la Figure 5:

- si l'APDU est uniquement authentifiée, elle contient l'étiquette d'APDU, le champ de longueur, le champ d'en-tête de sécurité, l'APDU d'origine et l'étiquette d'authentification;
- si l'APDU est chiffrée uniquement, elle contient l'étiquette d'APDU, le champ de longueur, le champ d'en-tête de sécurité et l'APDU DLMS chiffrée;
- si l'APDU est authentifiée et chiffrée, elle contient l'étiquette d'APDU, le champ de longueur, le champ d'en-tête de sécurité, l'APDU chiffrée et l'étiquette d'authentification.



Légende

Anglais	Français
Security context	Contexte de sécurité
xDLMS APDU	APDU xDLMS
Cipherring	Chiffrement
Ciphered xDLMS APDU	APDU xDLMS chiffrée
Authenticated APDU	APDU authentifiée
Tag	Étiquette
Len	Len
SH	SH
xDLMS APDU	APDU xDLMS
T	T
Encrypted APDU	APDU chiffrée
Tag	Étiquette
Len	Len
SH	SH
Encrypted xDLMS APDU	APDU xDLMS chiffrée
Authenticated and encrypted APDU	APDU authentifiée et chiffrée
Tag	Étiquette
Len	Len
SH	SH
Encrypted xDLMS APDU	APDU xDLMS chiffrée
T	T

Figure 5 – APDU xDLMS chiffrées

L'étiquette d'APDU identifie le type d'APDU. Voir l'Article 8.

Le champ de longueur identifie la longueur de la chaîne d'octets (incluant l'en-tête de sécurité, l'APDU xDLMS non chiffrée ou chiffrée et l'étiquette d'authentification).

Le champ Security Header, *SH*, inclut le champ Security Control, *SC*, concaténé vers la droite avec le champ Frame Counter, *FC*.

Le champ Security Control est présenté dans le Tableau 3, où:

- Bit 3...0: Security_Suite_Id; voir 5.4.4;
- Bit 4: Sous-champ « A » indique que l'APDU est authentifiée;
- Bit 5: Sous-champ « E » indique que l'APDU est chiffrée;
- Bit 6: Sous-champ Key_set 0 = Unicast;
1 = Broadcast; voir 5.4.7.2;
- Bit 7: Réservé, doit être défini sur 0

Tableau 3 – Champ Security Control

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3...0
Réservé, défini sur 0	Key_set	E	A	Security_Suite_Id

Le champ Frame Counter est un compteur interne maintenu par l'expéditeur et le destinataire.

Enfin, *T* est l'étiquette d'authentification, calculée par l'algorithme d'authentification.

5.4.7 Clés cryptographiques

5.4.7.1 Généralités

Une clé cryptographique est un paramètre utilisé en association avec un algorithme cryptographique qui détermine son fonctionnement de sorte qu'une entité connaissant la clé peut reproduire ou inverser l'opération, alors qu'une entité ne connaissant pas la clé en est incapable. Pour les besoins de DLMS/COSEM, voici quelques exemples d'opérations:

- transformation de données en texte brut en données en texte chiffré;
- transformation de données en texte chiffré en données en texte brut;
- calcul d'un code d'authentification à partir de données;
- vérification d'un code d'authentification à partir de données et d'un code d'authentification reçu.

5.4.7.2 Types de clés

Différents types de clés sont définis. Ils sont identifiés en fonction de leur classification, de leur fonction cryptographique, de leur utilisation et de leur portée.

Pour leur classification, une distinction est faite entre:

- les clés *symétriques*. Une clé symétrique est une clé cryptographique unique utilisée avec un algorithme de clé (symétrique) secret. Pour obtenir un aperçu des algorithmes de clés symétriques, voir E.3;
- les clés *publiques* et *privées*, utilisées avec les algorithmes cryptographiques de clés publiques. Pour obtenir un aperçu des algorithmes de clés asymétriques, voir E.4.

Pour les clés symétriques, une distinction est faite entre:

- *les clés d'authentification symétriques*, utilisées avec les algorithmes de clés symétriques pour garantir l'intégrité et la source des messages;
- *les clés de chiffrement de données symétriques*, utilisées avec les algorithmes de clés symétriques pour appliquer la protection de confidentialité aux informations, et
- *les clés d'encapsulation de clés symétriques*, utilisées pour chiffrer d'autres clés à l'aide d'algorithmes de clés symétriques. Les clés d'encapsulation de clés sont également connues sous le nom de clés de chiffrement de clés (Key Encrypting Keys ou KEK).

Pour leur utilisation, une distinction est faite entre:

- *les clés unicast*, utilisées pour sécuriser des communications unicast entre deux homologues;
- *les clés broadcast*, utilisées pour sécuriser des communications broadcast entre un client DLMS/COSEM et plusieurs serveurs DLMS/COSEM. Dans DLMS/COSEM, la diffusion peut être initiée uniquement par le client.

Pour leur portée, une distinction est faite entre les *clés dédiées* et les *clés globales*. Ces termes sont spécifiques à DLMS/COSEM:

- *une clé dédiée* est une clé de chiffrement qui peut être utilisée pour chiffrer des APDU xDLMS de l'établissement d'une AA entre un client et un serveur à la libération de l'AA. Par conséquent, sa durée de vie est la même que celle de l'AA. Elle est fournie durant l'établissement de l'AA et utilisée lors des transmissions ultérieures au sein de la même AA.
Lorsque l'AA est libérée, il convient que la clé dédiée soit supprimée par le serveur ainsi que par le client.
- *une clé globale* est une clé de chiffrement qui peut être (ré)utilisée pour chiffrer des APDU xDLMS, échangées entre le même client et le même serveur, dans plusieurs instances de l'AA.

Les clés de chiffrement (symétriques) suivantes peuvent être disponibles: clé unicast globale, clé broadcast globale, clé unicast dédiée.

Une clé d'authentification (symétrique) est toujours une clé globale qui peut être utilisée dans des messages unicast et broadcast.

En outre, des règles spécifiques à la suite de sécurité peuvent s'appliquer. Voir également le Tableau 4.

5.4.7.3 Gestion des clés

5.4.7.3.1 Architectures de système

NOTE Le paragraphe 5.4.7.3 fournit des instructions générales une implémentation sécurisée. La gestion du système de sécurité, y compris les clés de sécurité, est de la responsabilité de l'opérateur du système.

Aux fins du 5.4.7.3, deux architectures de systèmes généraux sont considérées:

- le client COSEM est le DCS central, qui échange des données directement avec les serveurs COSEM;
- le client COSEM est un concentrateur, qui échange des données d'une part avec les serveurs COSEM et d'autre part avec le DCS central. Dans ce cas, le concentrateur peut ou non également être un serveur COSEM. Le DCS central peut ou non être un client COSEM.

Pour la génération et la distribution des clés symétriques, la norme NIST SP 800-57:2006, 8.1.5.2 s'applique.

5.4.7.3.2 Clé maître

Une clé maître doit être présente sur chaque dispositif logique de serveur COSEM configuré dans le système. Cette clé permet d'encapsuler les clés globales. Voir 5.4.7.3.3.

NOTE 1 La méthode de génération et d'installation de la clé maître n'entre pas dans le domaine d'application de la présente spécification auxiliaire.

Un mécanisme sécurisé doit être mis en place pour garantir que le DCS de données central connaisse la clé maître de chaque dispositif logique de serveur COSEM dans le système.

NOTE 2 L'indication d'un tel mécanisme n'entre pas dans le domaine d'application de la présente spécification auxiliaire.

S'ils sont présents dans le système, les concentrateurs ne doivent pas connaître les clés maîtres.

Pour des raisons de sécurité, il convient que la durée de vie de la clé maître soit limitée. La méthode de remplacement des clés maîtres n'entre pas dans le domaine d'application de la présente norme. Pour obtenir des conseils, voir la norme NIST SP 800-57:2006, 8.2.3 intitulée « Key change function ».

5.4.7.3.3 Clés globales

Avant de pouvoir utiliser le chiffrement, les clés globales doivent être générées et fournies aux dispositifs logiques COSEM faisant partie du système.

Les clés globales doivent être générées par le DCS central.

La méthode de génération des clés globales n'entre pas dans le domaine d'application de la présente spécification auxiliaire. Pour des raisons de sécurité, il convient que leur durée de vie soit limitée.

Pour la livraison, les clés globales doivent être encapsulées à l'aide de l'algorithme d'encapsulation de clés AES-128 comme spécifié dans la RFC 3394, et de la clé maître comme clé de chiffrement de clés (KEK).

Lorsque le système fonctionne sans concentrateur, les clés globales doivent être fournies par le DCS central appelant la méthode `global_key_transfer` de l'objet d'interface `Security setup` sur le serveur COSEM avec la clé globale encapsulée définie comme paramètre. Voir la CEI 62056-6-2:—, 5.3.5.

Lorsque le système fonctionne avec un ou plusieurs concentrateurs, les clés globales doivent être fournies, encapsulées à l'aide de la clé maître, au concentrateur. La méthode `global_key_transfer` de l'objet d'interface `Security setup` du serveur COSEM doit être appelée par le concentrateur avec la clé globale encapsulée définie comme paramètre. Pour plus de détails, voir la CEI 62056-6-2:—, 5.3.5.

Dans tous les cas, la clé globale est encapsulée par la clé maître.

Il convient que le concentrateur ne connaisse pas les clés maîtres. Par conséquent, il convient qu'il ne puisse pas encapsuler/désencapsuler des clés globales.

Les clés globales doivent également être mises à disposition du/des concentrateur(s), le cas échéant, par le DCS central, d'une manière sécurisée.

NOTE L'indication d'un tel processus n'entre pas dans le domaine d'application de la présente partie de la CEI 62056.

5.4.7.3.4 Clés dédiées

Lorsque le système fonctionne sans concentrateur, les clés dédiées doivent être générées par le DCS central. Lorsque le système fonctionne avec un ou plusieurs concentrateurs, les clés dédiées doivent être générées par les concentrateurs.

Pour la livraison, elles doivent être incluses dans le champ de l'APDU xDLMS InitiateRequest prévu à cet effet, contenu dans le champ relatif aux informations sur l'utilisateur de l'APDU AARQ. L'APDU xDLMS InitiateRequest doit être authentifiée et chiffrée à l'aide de l'algorithme AES-GCM-128, de la clé de chiffrement unicast globale et, si elle est utilisée, de la clé d'authentification (voir 5.4.8.3.4.4). De la même manière, l'APDU xDLMS InitiateResponse, contenue dans le champ relatif aux informations sur l'utilisateur de l'APDU AARE, doit également être chiffrée et authentifiée. Voir également 5.2.

Les différents types de clés cryptographiques, leur utilisation et leur gestion sont récapitulés dans le Tableau 4.

Tableau 4 – Clés cryptographiques et leur gestion

Type de clé	Utilisation	Génération	Livraison	Emplacement
Clés symétriques				
Clé maître	Clé de chiffrement de clés (KEK) pour les clés globales	Non spécifiée	Non spécifiée	DCS central et serveur COSEM
Clé de chiffrement unicast globale	Chiffrement global d'APDU xDLMS unicast	DCS central	Encapsulée à l'aide de la clé maître, appel de la méthode global_key_transfer par le DCS central ou le concentrateur	DCS central (et concentrateur) et serveur COSEM
Clé de chiffrement broadcast globale	Chiffrement global d'APDU xDLMS broadcast	DCS central	Encapsulée à l'aide de la clé maître, appel de la méthode global_key_transfer par le DCS central ou le concentrateur	DCS central (et concentrateur) et serveur COSEM
Clé d'authentification (globale)	Authentification d'APDU xDLMS	DCS central	Encapsulée à l'aide de la clé maître, appel de la méthode global_key_transfer par le DCS central ou le concentrateur	DCS central (et concentrateur) et serveur COSEM
Clé de chiffrement (unicast) dédiée	Chiffrement dédié d'APDU xDLMS unicast	DCS central ou concentrateur	Transportée dans le cadre de l'APDU xDLMS InitiateRequest, qui est chiffrée et authentifiée à l'aide de l'algorithme AES-GCM-128, de la clé de chiffrement unicast globale et de la clé d'authentification	DCS central (ou concentrateur) et serveur COSEM (durant la durée de vie de l'AA)
NOTE Si des alternatives sont proposées, la première alternative est valide pour les systèmes n'utilisant pas de concentrateur et la deuxième, pour les systèmes utilisant des concentrateurs.				

5.4.8 Mode de fonctionnement Galois/Counter (GCM)

5.4.8.1 Généralités

NOTE Le texte suivant est issu de la publication NIST SP 800-38D:2007, Article 3.

GCM est un algorithme de chiffrement authentifié avec des données associées. GCM est construit à partir d'un chiffrement de bloc à clé symétrique approuvé, avec une taille de bloc de 128 bits, par exemple, l'algorithme AES. Voir la norme FIPS PUB 197. Par conséquent, GCM est un mode de fonctionnement de l'algorithme AES.

GCM offre la garantie de la confidentialité des données à l'aide d'une variation du mode de fonctionnement Counter pour le chiffrement.

GCM offre la garantie de l'authenticité des données confidentielles (jusqu'à environ 64 gigaoctets par appel) à l'aide d'une fonction de hachage universelle qui est définie dans un champ Galois binaire (c'est-à-dire défini). GCM peut également offrir la garantie de l'authentification des données supplémentaires (longueur presque illimitée par appel) qui ne sont pas chiffrées.

Si les entrées GCM sont limitées aux données qui ne sont pas chiffrées, la spécialisation résultante de GCM, appelée GMAC, est simplement un mode d'authentification appliqué aux données entrantes.

GCM offre une garantie d'authentification supérieure à celle d'une somme de contrôle ou d'un code de détection d'erreur (non cryptographique). GCM peut en particulier détecter les modifications accidentelles des données et les modifications intentionnelles, non autorisées.

Dans DLMS/COSEM, il est également possible d'utiliser GCM pour offrir uniquement la confidentialité. Dans ce cas, les étiquettes d'authentification ne sont simplement pas calculées ni vérifiées.

Pour plus d'informations, voir la publication NIST SP 800-38D.

5.4.8.2 Définitions, abréviations, symboles et notation pertinents pour le mode GCM

Terme/Abréviation	Symbole	Définition
Données authentifiées supplémentaires (AAD)	A	Données entrantes pour la fonction de chiffrement authentifié, qui sont authentifiées, mais pas chiffrées. Elles sont également connues sous le nom de Données associées.
Déchiffrement authentifié		Fonction de GCM dans laquelle le texte chiffré est déchiffré en texte brut, et l'authenticité du texte chiffré ainsi que les AAD sont vérifiées.
Chiffrement authentifié		Fonction de GCM dans laquelle le texte brut est chiffré en texte chiffré, et une étiquette d'authentification est générée pour le texte chiffré ainsi que les AAD.
Authentication Key (Clé d'authentification)	AK	Partie des AAD
Chiffrement de bloc		Famille paramétrée de permutations sur des chaînes de bits d'une longueur fixe. Le paramètre qui détermine la permutation est une chaîne de bits appelée la clé.
Texte chiffré	C	Forme chiffrée du texte brut
	EK	Clé de chiffrement de bloc
Champ fixe		Dans la construction déterministe des IV, champ qui identifie le dispositif ou le contexte de l'instance de la fonction de chiffrement authentifié.
	FC	Compteur de trames
Fraîche		Pour une clé récemment générée, propriété qui consiste à être différente de toutes les clés précédemment utilisées.
GCM		Mode Galois/Counter
Initialisation Vector (Vecteur d'initialisation)	IV	Valeur unique associée à l'appel d'un chiffrement authentifié pour un texte brut spécifique et des AAD.
Champ d'appel		Dans la construction déterministe des IV, champ qui identifie les ensembles d'entrées dans la fonction de chiffrement authentifié dans un dispositif ou un contexte spécifique. Pour les besoins de cette spécification auxiliaire, le champ d'appel est le compteur de trames.
Clé		Paramètre du chiffrement de bloc qui détermine la sélection de la fonction de transmission de chiffrement dans la famille de permutations.
	KEK	Clé de chiffrement de clés

Terme/Abréviation	Symbole	Définition
	$len(X)$	Longueur en bits de la chaîne de bits X
	$LEN(X)$	Longueur en octets de la chaîne d'octets X
Valeur unique		Valeur utilisée une seule fois dans un contexte spécifique
Texte brut	P	Données entrantes pour la fonction de chiffrement authentifié, qui sont authentifiées et chiffrées.
	SC	Octet de contrôle de sécurité
	SH	En-tête de sécurité; concaténation de l'octet de contrôle de sécurité (SC) et du compteur de trames (FC). $SH = SC \parallel FC$.
Titre système	$Sys-T$	Identifiant unique du système
Étiquette d'authentification	T	Somme de contrôle cryptographique appliquée à des données, qui est conçue pour révéler les erreurs accidentelles et les modifications intentionnelles des données.
	t	Longueur en bits de l'étiquette d'authentification NOTE Cette valeur est identique à $len(T)$.
	$X \parallel Y$	Concaténation de deux chaînes, X et Y .

5.4.8.3 Éléments de GCM

5.4.8.3.1 Chiffrement de bloc et clés

Pour les besoins de la présente spécification auxiliaire, la norme de chiffrement AES a été sélectionnée. GCM utilise la fonction de transmission de chiffrement et non celle d'inversion de chiffrement.

GCM utilise une clé unique, la clé de chiffrement de bloc (abrégée en clé). Sa taille doit être de 128 bits. Pour plus de sécurité, une clé d'authentification est également indiquée dans DLMS/COSEM. Elle fait partie des AAD. La clé de chiffrement de bloc est symbolisée par EK et la clé d'authentification, par AK .

5.4.8.3.2 Fonctions de GCM

5.4.8.3.2.1 Généralités

NOTE Le texte suivant est basé sur la norme NIST SP 800-38D:2007, 5.2.

Les deux fonctions qui constituent GCM sont appelées chiffrement authentifié et déchiffrement authentifié. La fonction de chiffrement authentifié chiffre les données confidentielles et calcule une étiquette d'authentification pour les données confidentielles ainsi que pour toutes les données supplémentaires, non confidentielles. La fonction de déchiffrement authentifié déchiffre les données confidentielles en fonction de la vérification de l'étiquette.

Lorsque l'entrée est limitée à des données non confidentielles, la variante résultant de GCM est appelée GMAC. Pour GMAC, les fonctions de chiffrement et de déchiffrement authentifiés deviennent les fonctions de génération et de vérification d'une étiquette d'authentification pour les données non confidentielles.

Enfin, si l'authentification n'est pas requise, la fonction de chiffrement authentifié chiffre les données confidentielles, mais l'étiquette d'authentification n'est pas calculée. La fonction de déchiffrement authentifié déchiffre les données confidentielles, mais aucune étiquette d'authentification n'est calculée ni vérifiée.

5.4.8.3.2.2 Chiffrement authentifié – données entrantes et sortantes

NOTE Le texte suivant est basé sur la norme NIST SP 800-38D:2007, 5.2.1.1, et la norme RFC 4106:2005, 2.1.

L'opération de chiffrement authentifié comprend quatre entrées, chacune d'entre elles devant être une chaîne d'octets:

- du texte brut, symbolisé par P ;
- des données AAD, symbolisées par A . Elles doivent inclure, parmi tous les éléments, la clé d'authentification, symbolisée par AK .
- une clé de chiffrement de bloc secrète, symbolisée par EK ;
- un vecteur d'initialisation, symbolisé par IV .

Le texte brut et les AAD constituent les deux catégories de données protégées par GCM. GCM protège l'authenticité du texte brut et des AAD. Il protège également la confidentialité du texte brut, alors que les AAD sont conservées en clair.

L' IV est une variable unique, c'est-à-dire une valeur qui est unique dans le contexte (de sécurité) spécifié, qui détermine un appel de la fonction de chiffrement authentifié pour les données entrantes à protéger. Voir 5.4.8.3.4.5.

Il existe deux sorties:

- du texte chiffré, symbolisé par C , dont la longueur est exactement la même que celle du texte brut P ;
- une étiquette d'authentification, symbolisée par T .

5.4.8.3.2.3 Déchiffrement authentifié – données entrantes et sortantes

L'opération de déchiffrement authentifié comprend cinq entrées:

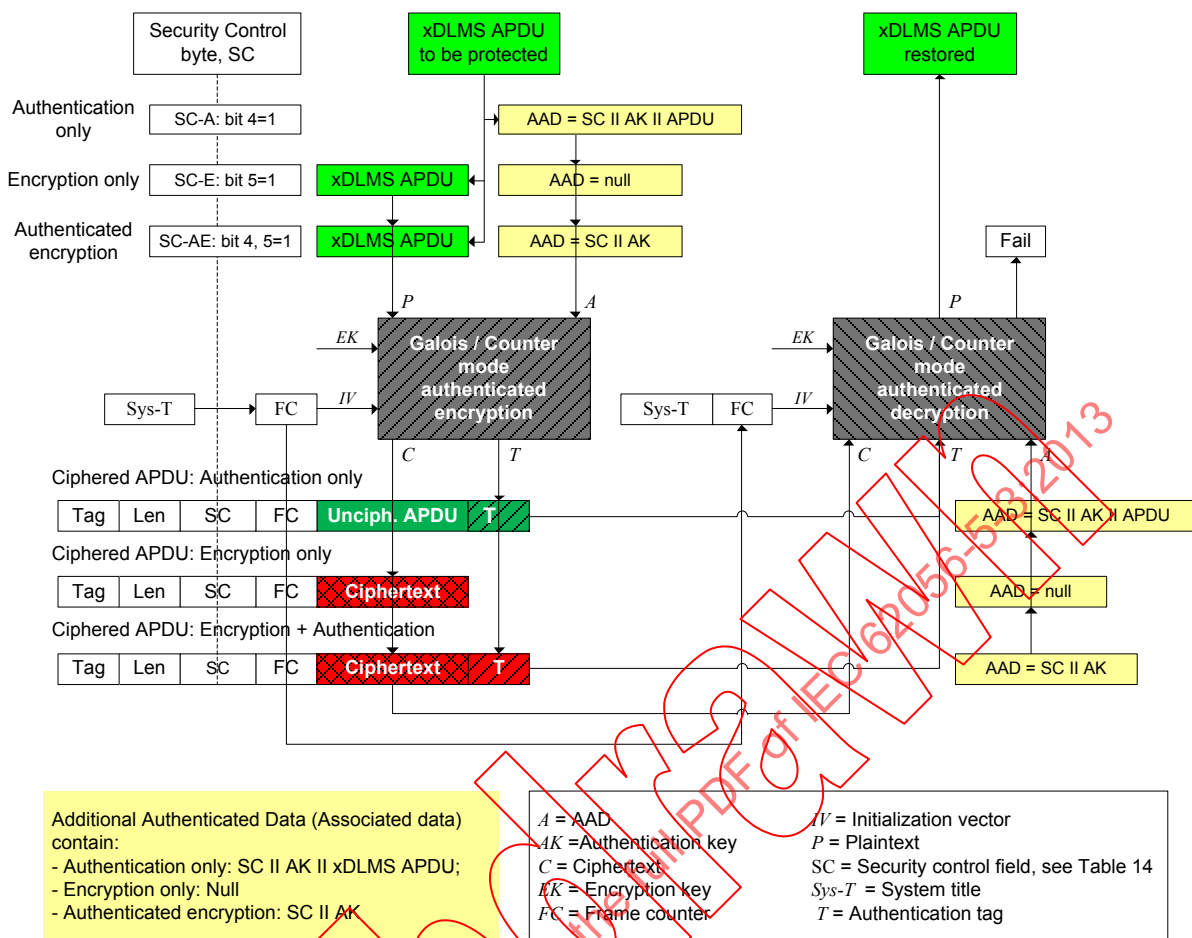
- la clé de chiffrement de bloc secrète, symbolisée par EK ;
- le vecteur d'initialisation, symbolisé par IV ;
- le texte chiffré, symbolisé par C ;
- les données AAD, symbolisées par A ;
- l'étiquette d'authentification, symbolisée par T .

La sortie est l'une des suivantes.

- le texte brut P qui correspond au texte chiffré C ou
- un code d'erreur spécial, symbolisé par $FAIL$ dans le présent document.

La sortie P indique que T est l'étiquette d'authentification correcte pour IV , A et C . Sinon, la sortie est $FAIL$.

Les fonctions de chiffrement et de déchiffrement authentifiés, leurs entrées et leurs sorties, ainsi que la structure des APDU sécurisées sont présentées dans la Figure 6 ci-dessous.



IEC 1117/13

Légende

Anglais	Français
Security Control byte	Octet de contrôle de sécurité
xDLMS APDU to be protected	APDU xDLMS à protéger
xDLMS APDU restored	APDU xDLMS restaurée
Authentication only	Authentification uniquement
SC-A	SC-A
AAD = SC-A AK APDU	AAD = SC-A AK APDU
Encryption only	Chiffrement uniquement
SC-E	SC-E
xDLMS APDU	APDU xDLMS
AAD = null	AAD = null
Authenticated encryption	Chiffrement authentifié
SC-AE	SC-AE
AAD = SC-AE AK	AAD = SC-AE AK
Fail	Échec
Sys-T	Sys-T
FC	FC
IV	IV
EK	EK
P	P

Anglais	Français
A	A
C	C
T	T
Galois/Counter Mode authenticated encryption	Chiffrement authentifié à l'aide de GCM
FC	FC
IV	IV
EK	EK
P	P
A	A
C	C
T	T
Galois/Counter Mode authenticated decryption	Déchiffrement authentifié à l'aide de GCM
Ciphered APDU: Authentication only	APDU chiffrée: authentification uniquement
Tag	Étiquette
Len	Len
SC-A	SC-A
FC	FC
Unciph. APDU	APDU non chiffrée
AAD = SC-A AK APDU	AAD = SC-A AK APDU
Ciphered APDU: Encryption only	APDU chiffrée: chiffrement uniquement
Tag	Étiquette
Len	Len
SC-E	SC-E
FC	FC
Ciphertext	Texte chiffré
AAD = null	AAD = null
Ciphered APDU: Encryption + Authentication	APDU chiffrée: chiffrement + authentification
Tag	Étiquette
Len	Len
SC-AE	SC-AE
FC	FC
Ciphertext	Texte chiffré
AAD = SC-AE AK	AAD = SC-AE AK
Additional Authenticated Data (Associated Data) contain: - Authentication only: SC-A AK xDLMS APDU; - Encryption only: null - Authenticated encryption: SC-AE AK	Les AAD (données associées) contiennent: - Authentification uniquement: SC-A AK xDLMS APDU ; - Chiffrement uniquement: null - Chiffrement authentifié: SC-AE AK
A = AAD	A = AAD
AK = Authentication Key	AK = Clé d'authentification
C = Ciphertext	C = Texte chiffré
EK = Encryption Key	EK = Clé de chiffrement
FC = Frame counter	FC = Compteur de trames
IV = Initialization vector	IV = Vecteur d'initialisation
P = Plaintext	P = Texte brut

Anglais	Français
SC = Security control byte: SC-A: Authentication only SC-E: Encryption only SC-AE: Authenticated encryption	SC = Octet de contrôle de sécurité SC-A: Authentification uniquement SC-E: Chiffrement uniquement SC-AE: Chiffrement authentifié
Sys-T = System Title	Sys-T = Titre système
T = Authentication Tag	T = Étiquette d'authentification

Figure 6 – Protection cryptographique des APDU xDLMS à l'aide de GCM

5.4.8.3.3 En-tête de sécurité avec GCM

L'en-tête de sécurité doit être constitué comme indiqué en 5.4.6, à savoir de `security_suite_id` = 0 et des valeurs appropriées pour les sous-champs `Key_set`, `E` et `A`.

5.4.8.3.4 Paramètres de GCM et éléments de données

5.4.8.3.4.1 Longueur de l'étiquette d'authentification

La longueur en bits de l'étiquette d'authentification, symbolisée par t , est un paramètre de sécurité. Sa valeur doit être de 96 bits.

5.4.8.3.4.2 Texte brut et AAD

Le texte brut, symbolisé par P et les AAD, symbolisées par A , dépendent du type de protection. Ils sont présentés dans le Tableau 5, où SC désigne l'octet de contrôle de sécurité, AK , la clé d'authentification et M , le message, c'est-à-dire, l'APDU xDLMS à protéger.

Les longueurs en bits de P et de A doivent remplir les exigences suivantes:

- $\text{len}(P) < 2^{39}-256$;
- $\text{len}(A) < 2^{64}-1$;
- les longueurs en bits de P et de A doivent être des multiples de 8, de sorte que ces valeurs sont des chaînes d'octets.

Tableau 5 – Texte brut et AAD

Contrôle de sécurité, SC		Security	P	A
Champ E	Champ A			
0	0	Rien	–	–
0	1	Authentifiée uniquement	–	$SC \parallel AK \parallel M$
1	0	Chiffrée uniquement	M	–
1	1	Chiffrée et authentifiée	M	$SC \parallel AK$

5.4.8.3.4.3 Clé de chiffrement de bloc, EK

La taille de la clé de chiffrement de bloc, symbolisée par EK , doit être de 128 bits (16 octets). $\text{len}(EK) = 128$.

La clé doit être générée uniformément et de manière aléatoire, ou presque uniformément et de manière aléatoire. En d'autres termes, chaque clé possible a (quasiment) autant de chances d'être générée. Par conséquent, la clé est "fraîche", c'est-à-dire qu'il y a de fortes probabilités qu'elle ne soit égale à aucune autre clé. La clé doit être secrète et utilisée exclusivement pour GCM avec le chiffrement de bloc AES choisi. Les autres exigences relatives à l'établissement et à la gestion de clés sont abordées dans la norme NIST SP 800-38D:2007, 8.1.

5.4.8.3.4.4 Clé d'authentification, AK

Même si elle n'est pas requise par GCM, une clé d'authentification, symbolisée par AK , est indiquée pour accroître la sécurité dans DLMS/COSEM. Pour sa longueur et sa génération, les mêmes règles que pour la clé de chiffrement s'appliquent. La clé d'authentification doit faire partie des AAD. Si une clé d'authentification a été transférée, elle doit être utilisée jusqu'à ce qu'une clé d'authentification nulle soit transférée.

5.4.8.3.4.5 Vecteur d'initialisation, IV

Pour la construction du vecteur d'initialisation, IV , la construction déterministe indiquée dans la norme NIST SP 800-38D:2007, 8.2.1, doit être utilisée.

Dans la construction déterministe, l' IV est la concaténation de deux champs, appelés le champ fixe et le champ d'appel. Le champ fixe doit identifier le dispositif physique ou, de manière plus générale, le contexte (de sécurité) pour l'instance de la fonction de chiffrement authentifié. Le champ d'appel doit identifier les ensembles d'entrées dans la fonction de chiffrement authentifié pour ce dispositif spécifique.

Pour n'importe quelle clé donnée, deux dispositifs physiques distincts ne doivent pas partager le même champ fixe et deux ensembles distincts d'entrées pour un seul dispositif ne doivent pas partager le même champ d'appel.

La longueur de l' IV doit être de 96 bits (12 octets): $len(IV) = 96$. Dans cette formule:

- les 64 bits (8 octets) de départ (les plus à gauche) doivent contenir le champ fixe. Ils doivent contenir le titre du système. Voir 5.4.8.3.4.6;
- les 32 bits de fin (les plus à droite) doivent contenir le champ d'appel. Voir 5.4.8.3.4.7.

La longueur en bits du champ fixe limite à 2^{64} le nombre de dispositifs physiques distincts qui peuvent implémenter la fonction de chiffrement authentifié pour la clé donnée. La longueur en bits du champ d'appel limite à 2^{32} le nombre d'appels de la fonction de chiffrement authentifié, quels que soient les ensembles d'entrées, sans violation de l'exigence d'unicité.

5.4.8.3.4.6 Titre système

Chaque dispositif physique du système, y compris les compteurs, les concentrateurs (le cas échéant) et le DCS doivent être identifiés de manière unique par leur titre système. Le titre système, symbolisé par $Sys-T$, doit être construit de la manière suivante:

- sa longueur doit être de 8 octets;
- les octets de départ (les 3 les plus à gauche) doivent contenir l'ID à trois lettres du fabricant⁶. Il est identique à celui utilisé dans les octets de départ du LDN des dispositifs logiques du serveur;

L'ID à trois lettres du fabricant doit être utilisé pour identifier tous systèmes clients et serveurs.

NOTE 1 Il peut être obtenu auprès de la DLMS UA.

- les octets de fin (les 5 les plus à droite) et l'ID du fabricant doivent garantir l'unicité du titre système.

NOTE 2 Ces 5 octets peuvent être par exemple déduits des 12 derniers chiffres du numéro de fabrication, jusqu'à 999 999 999 999. Cette valeur est convertie en 0xE8D4A50FFF. Les valeurs supérieures, jusqu'à 0xFFFFFFFF (valeur décimale: 1 099 511 627 775), peuvent également être utilisées, mais ne peuvent pas être mappées aux 12 derniers chiffres du numéro de fabrication.

⁶ Géré par DLMS-UA agissant comme autorité d'enregistrement; identique à celui utilisé dans le LDN.

Des spécifications auxiliaires spécifiques à un projet donné peuvent indiquer une structure différente pour le titre système. Cependant, dans tous les cas, la longueur du titre système doit être de 8 octets (et si nécessaire, rallongée vers la droite) et ce dernier doit être unique. Les informations détaillées doivent être indiquées par l'autorité de dénomination désignée comme telle pour le projet.

Avant de pouvoir utiliser les algorithmes de sécurité de message, qui requièrent une application chiffrée, les partenaires doivent échanger des titres système. Les possibilités suivantes sont disponibles:

- lorsque le profil CPL S-FSK est utilisé, les titres système sont échangés durant le processus d'enregistrement à l'aide du protocole CIASE. Voir la CEI 62056-8-3;
- dans tous les profils de communication, les titres système sont échangés durant l'établissement de l'AA avec un contexte chiffré à l'aide du service COSEM-OPEN (voir 6.2) contenu dans les APDU AARQ/AARE;

Si les titres système envoyés/reçus durant l'établissement d'une AA ne sont pas identiques à ceux échangés durant le processus d'enregistrement CIASE, l'AA doit être rejetée.

- en écrivant l'attribut `client_system_title` et en lisant l'attribut `server_system_title` des objets Security setup. Voir la CEI 62056-6-2:—, 5.3.5.

NOTE 3 Pour la communication broadcast, seul le client COSEM l'envoie au serveur COSEM.

5.4.8.3.4.7 Champ d'appel

Le champ d'appel doit être un compteur d'entiers (compteur de trames).

5.4.8.3.5 Exemple

L'exemple suivant (voir Tableau 6) illustre le processus de création d'APDU xDLMS chiffrées lors de l'application de l'authentification seule, du chiffrement seul et du chiffrement authentifié.

Tableau 6 – Exemple d'APDU chiffrées

	X	Contenu			LEN (X) octets	Len (X) bits
Matériel de sécurité						
Suite de sécurité		GCM-AES-128				
Titre système	Sys-T	4D4D4D0000BC614E (ici, les cinq derniers octets contiennent le numéro de fabrication au format hexadécimal)			8	64
Compteur de trames	FC	01234567			4	32
Vecteur d'initialisation	IV	Sys-T FC			12	96
		4D4D4D0000BC614E01234567				
Clé de chiffrement de bloc (globale)	EK	000102030405060708090A0B0C0D0E0F			16	128
Clé d'authentification	AK	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF			16	128
Sécurité appliquée		Authentification	Chiffrement	Chiffrement authentifié		
Octet de contrôle de sécurité (avec clé unicast)	SC	SC-A	SC-E	SC-AE	1	8
		10	20	30		
En-tête de sécurité	SH	SH = SC-A FC	SH = SC-E FC	SH = SC-AE FC		
		1001234567	2001234567	3001234567	5	40
Entrées		Authentification	Chiffrement	Chiffrement authentifié		
APDU xDLMS à protéger	APDU	C001000008000000100000FF0200 (Get-request, attribut 2 de l'objet Clock)			13	104
Texte brut	P	Null	C0010000080000001 0000FF0200	C0010000080000001 0000FF0200	13	104
Données associées	A	SC AK APDU	–	SC AK		
Données associées – Authentification	A-A	10D0D1D2D3D4D5D6 D7D8D9DADBDCDDDE DFC0010000080000 010000FF0200	–	–	30	240
Données associées – Chiffrement	A-E	–	–	–	0	0
Données associées – Chiffrement authentifié	A-AE	–	–	30D0D1D2D3D4D5D6 D7D8D9DADBDCDDDE DF	17	136
Sorties		Authentification	Chiffrement	Chiffrement authentifié		
Texte chiffré	C	NULL	411312FF935A4756 6827C467BC	411312FF935A4756 6827C467BC	13	104
Étiquette d'authentification	T	06725D910F9221D2 63877516	–	7D825C3BE4A77C3F CC056B6B	12	96
APDU chiffrée complète		TAG LEN SH APDU T	TAG LEN SH C	TAG LEN SH C T	–	–
APDU authentifiée		C81E1001234567C0 01000008000000100 00FF020006725D91 0F9221D263877516	–	–	32	256

	<i>X</i>	Contenu			<i>LEN</i> (<i>X</i>) octets	<i>Len</i> (<i>X</i>) bits
APDU chiffrée		–	C812200123456741 1312FF935A475668 27C467BC	–	20	160
APDU chiffrée et authentifiée		–	–	C81E300123456741 1312FF935A475668 27C467BC7D825C3B E4A77C3FCC056B6B	32	256

5.4.8.4 Authentification de l'homologue HLS avec GMAC (authentication_mechanism_id(5))

NOTE Voir également 5.3.4.

Dans le cas d'authentication_mechanism_id(5), le processus est le suivant:

- Pass 1 et Pass 2 doivent être identiques à ceux décrits en 5.3.4;
- dans Pass 3 et 4, au lieu du secret HLS (contenu dans les objets Association LN/Association SN), la clé de chiffrement unicast globale et la clé d'authentification (le cas échéant) sont utilisées. Celles-ci sont transférées au serveur à l'aide de la méthode global_key_transfer de la classe Security setup. Voir la CEI 62056-6-2:—, 5.3.5;
- dans Pass 3, le client traite le défi StoC. L'étiquette d'authentification, *T*, est calculée à l'aide de GMAC sur *SC* || *AK* || StoC. Le résultat $f(\text{StoC}) = SC \parallel FC \parallel T$ est renvoyé au serveur. Le serveur vérifie si $f(\text{StoC})$ est le résultat du traitement correct et, le cas échéant, accepte l'authentification du client;
- dans Pass 4: si le client est authentifié, le serveur traite la demande CtoS de la même manière que décrit dans Pass 3. L'étiquette d'authentification *T* est calculée à l'aide de GMAC sur *SC* || *AK* || CtoS. Le résultat $f(\text{CtoS}) = SC \parallel FC \parallel T$ est renvoyé au serveur. Le client vérifie si $f(\text{CtoS})$ est le résultat du traitement correct et, le cas échéant, accepte l'authentification du serveur.

Un exemple est donné dans le Tableau 7 ci-dessous.

Tableau 7 – Exemple HLS avec GMAC

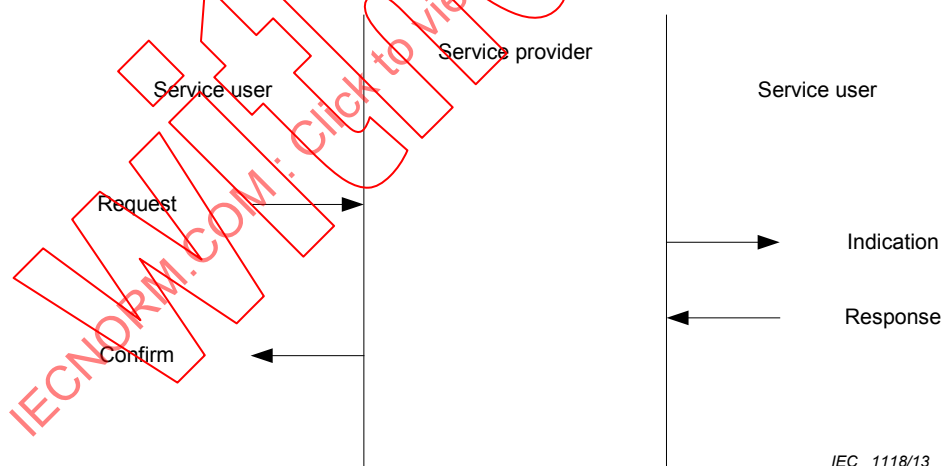
Matériel de sécurité	<i>X</i>	Contenu		<i>LEN(X)</i> octets	<i>len(X)</i> bits
Suite de sécurité		GCM-AES-128			
Titre système	<i>Sys-T</i>	Client	Serveur		
		4D4D4D0000000001	4D4D4D0000BC614E		
		(ici, les cinq derniers octets contiennent le numéro de fabrication au format hexadécimal)		8	64
Compteur de trames	<i>FC</i>	00000001	01234567	4	32
Vecteur d'initialisation	<i>IV</i>	<i>Sys-T</i> <i>FC</i>		12	96
		Client	Serveur		
		4D4D4D0000000001 00000001	4D4D4D0000BC614E 01234567		
Clé de chiffrement de bloc (globale)	<i>EK</i>	000102030405060708090A0B0C0D0E0F		16	128
Clé d'authentification	<i>AK</i>	D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF		16	128
Contrôle de sécurité	<i>SC</i>	10		1	8
		Pass 1: Le client envoie un défi au serveur			

Matériel de sécurité	X	Contenu	LEN(X) octets	len(X) bits
CtoS		4B35366956616759 "K56iVagY"	8	64
		Pass 2: Le serveur envoie un défi au client		
StoC		503677524A323146 "P6wRJ21F"	8	64
		Pass 3: Le client traite le StoC		
SC II AK II StoC		10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF503677524A323146		
T (SC II AK II StoC)		1A52FE7DD3E72748973C1E28	12	96
f(StoC)		10000000011A52FE7DD3E72748973C1E28	17	136
		Pass 4: Le serveur traite le CtoS		
SC II AK II CtoS		10D0D1D2D3D4D5D6D7D8D9DADBDCDDDEDF4B35366956616759		
T (SC II AK II CtoS)		FE1466AFB3DBCD4F9389E2B7	12	96
f(CtoS)		1001234567FE1466AFB3DBCD4F9389E2B7	17	136

6 Spécification de service de la couche application COSEM

6.1 Primitives de service et paramètres

En général, les services d'une couche (ou d'une sous-couche) sont des capacités offertes à un utilisateur dans la prochaine couche supérieure (ou sous-couche). Afin de fournir ses services, une couche base ses fonctions sur les services qu'elle requiert de la couche inférieure précédente. La Figure 7 illustre cette notion de hiérarchie de service et présente la relation entre deux utilisateurs N correspondants et leurs entités de protocole homologue de couche N.



Légende

Anglais	Français
Service User	Usager
Service Provider	Fournisseur de services
Service User	Usager
Request	Demande
Indication	Indication
Response	Réponse
Confirm	Confirmation

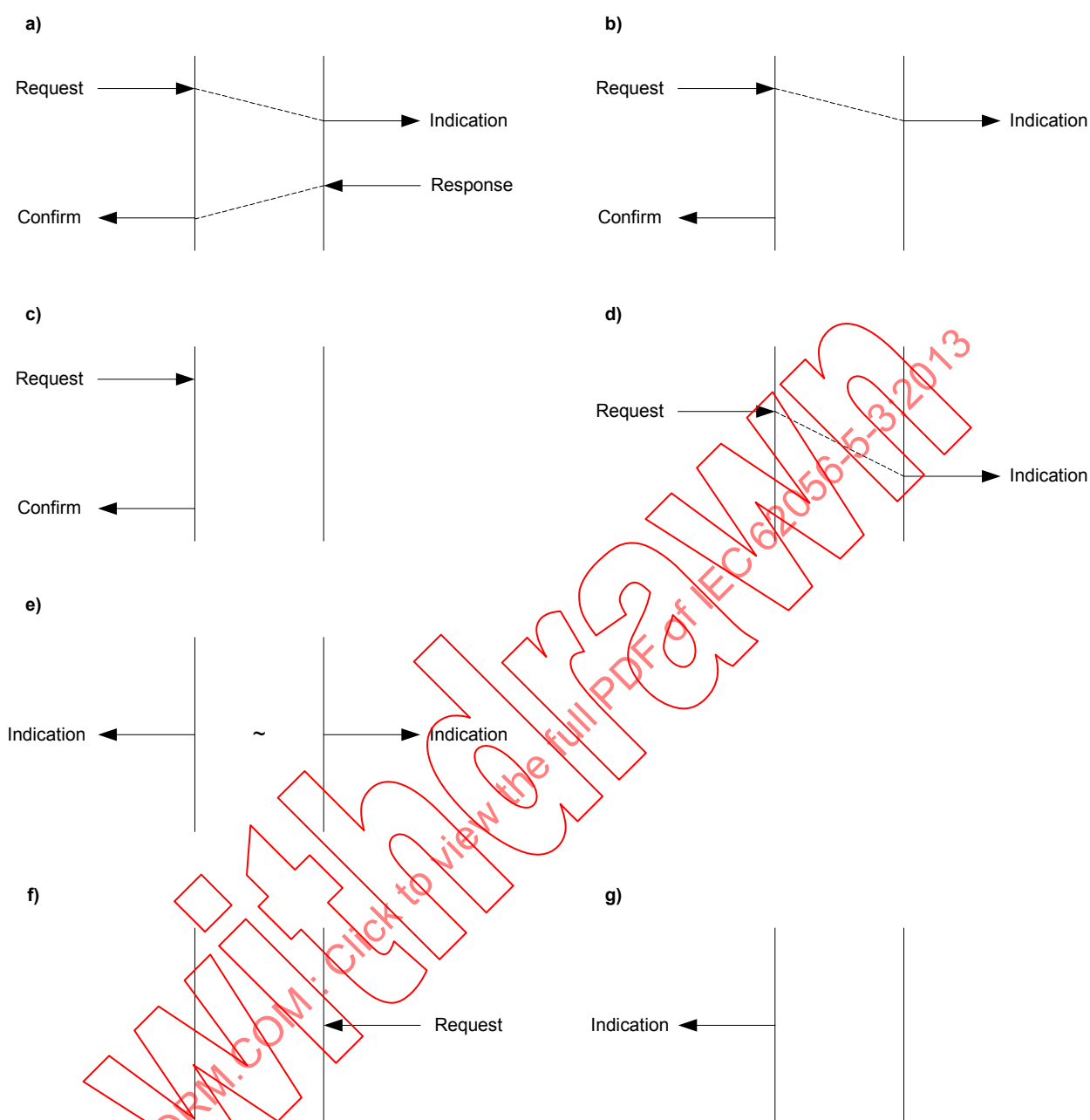
Figure 7 – Primitives de service

Les services sont indiqués en décrivant le flux d'informations entre l'utilisateur N et la couche N. Ce flux d'informations est modélisé par des événements discrets et instantanés, qui caractérisent la fourniture d'un service. Chaque événement consiste en la transmission d'une primitive de service d'une couche à l'autre via un point d'accès aux services de la couche N associé à un utilisateur N. Les primitives de service transportent les informations requises en fournissant un service spécifique. Ces primitives de service sont abstraites du fait qu'elles n'indiquent que le service fourni et non les moyens par lesquels le service est fourni. Cette définition de service est indépendante de toute implémentation d'interface spécifique.

Les services sont indiqués en décrivant les primitives de service et les paramètres qui caractérisent chaque service. Un service peut avoir une ou plusieurs primitives associées, qui constituent l'activité associée au service spécifique. Chaque primitive de service peut comprendre zéro paramètre ou plus transportant les informations requises pour fournir le service. Les primitives sont de quatre types génériques:

- **REQUETE:** La primitive de requête est transmise de l'utilisateur N à la couche N pour demander l'initiation d'un service;
- **INDICATION:** La primitive d'indication est transmise de la couche N à l'utilisateur N pour indiquer un événement de couche N interne significatif pour l'utilisateur N. Cet événement peut être logiquement associé à une demande de service distant ou peut être provoqué par un événement interne à la couche N;
- **RÉPONSE:** La primitive de réponse est transmise de l'utilisateur N à la couche N pour exécuter une procédure précédemment appelée par une primitive d'indication;
- **CONFIRMATION:** La primitive de confirmation est transmise de la couche N à l'utilisateur N pour transporter les résultats d'une ou de plusieurs demandes de service précédentes associées.

Les relations possibles entre les types de primitives sont illustrées par les diagrammes de séquences temporelles présentés dans la Figure 8. La figure indique également la relation logique des types de primitives. Les types de primitives qui se produisent plus tôt dans le temps et sont connectés par des lignes pointillées dans les diagrammes sont les antécédents logiques des types de primitives ultérieures.



Légende

Anglais	Français
Request	Requête
Indication	Indication
Response	Réponse
Confirm	Confirmation

Figure 8 – Diagrammes de séquences temporelles

Les paramètres de service des primitives de service de l'AL COSEM sont présentés sous forme de tableau. Chaque tableau comprend de deux à cinq colonnes qui décrivent les primitives de service et leurs paramètres. Dans chaque tableau, un paramètre (ou une partie de celui-ci) est répertorié sur chaque ligne. Dans les colonnes des primitives de service appropriées, un code est utilisé pour indiquer le type d'utilisation du paramètre. Les codes utilisés sont répertoriés dans le Tableau 8.

Certains paramètres peuvent contenir des sous-paramètres. Ceux-ci sont indiqués par l'intermédiaire d'une étiquette (M, U, S ou C) apposée sur les paramètres. Les sous-paramètres sont indentés sous chaque paramètre. La présence de sous-paramètres dépend toujours de celle du paramètre sous lequel ils apparaissent. Par exemple, un paramètre facultatif peut comprendre des sous-paramètres. Si le paramètre n'est pas indiqué, aucun sous-paramètre ne peut être indiqué.

Tableau 8 – Codes des paramètres de service de l'AL

M	Paramètre obligatoire pour la primitive
U	Option de l'utilisateur; ce paramètre peut ou non être indiqué, en fonction de l'utilisation dynamique par l'utilisateur de l'ASE.
S	Paramètre sélectionné parmi les autres paramètres S en tant que réponse interne de l'environnement du serveur de l'ASE.
C	Paramètre dépendant d'autres paramètres ou de l'environnement de l'utilisateur de l'ASE.
(-)	Paramètre jamais spécifié.
=	Le code « (=) » suivant le code M, U, S ou C indique que le paramètre équivaut sur le plan sémantique au paramètre de la primitive de service se trouvant immédiatement à sa gauche dans le tableau. Par exemple, un code « M(=) » dans la colonne de la primitive de service .indication et un « M » dans la colonne de la primitive de service .request indiquent que le paramètre dans la primitive .indication équivaut sur le plan sémantique à celui dans la primitive .request.

Dans la présente norme, les règles suivantes sont observées en ce qui concerne la dénomination de termes:

- le nom des services ACSE et des services de transfert de données utilisant le référencement LN est écrit en majuscules. Exemples: COSEM-OPEN, GET;
- le nom des services de transfert de données utilisant le référencement SN est écrit en casse de titre. Exemples: Read, Write;
- la notation mixte est utilisée dans les cas suivants: EventNotification, TriggerEventNotificationSending, UnconfirmedWrite, InformationReport;
- les types de primitives de service LN peuvent être mentionnés sous deux formes alternatives. Exemples: « primitive de service GET.request de type Request_Type == NORMAL » ou « primitive de service GET-REQUEST-NORMAL »;
- les éléments de nom des paramètres de service sont écrits en majuscules et associés par un trait de soulignement afin de ne constituer qu'une seule entité: par exemple, Protocol_Connection_Parameters et COSEM_Attribute_Descriptor;
- lorsque le même paramètre peut revenir plusieurs fois, le paramètre est répété entre accolades. Exemple: Data { Data };
- dans les spécifications de service de transfert de données, les paramètres utilisés avec le transfert de bloc uniquement sont affichés en gras. Exemple: **DataBlock_G**;
- la référence directe à un paramètre de service utilise les majuscules, alors que la référence indirecte (non spécifique) utilise le texte normal sans trait de soulignement. Exemple de référence directe: "Le paramètre COSEM_Attribute_Descriptor se rapporte à un attribut d'objet COSEM." Exemple de référence indirecte (non spécifique): "Une primitive de service GET-REQUEST-NORMAL contient un descripteur d'attribut COSEM unique.";
- les noms des APDU de transfert de données COSEM utilisant le référencement LN sont écrits en majuscules et associés par un tiret afin de ne constituer qu'une seule entité. Exemple: Get-Request-Normal;
- les noms des APDU de transfert de données COSEM utilisant le référencement SN utilisent la notation mixte. Exemple: ReadRequest.

6.2 Service COSEM-OPEN

NOTE La description a été modifiée pour couvrir les nouvelles fonctionnalités de sécurité du transport des données. L'ordre des paramètres de service a été aligné avec celui des champs des APDU AARQ/AARE.

Fonction

La fonction du service COSEM-OPEN est d'établir une AA entre des AP COSEM homologues. Il utilise le service A-ASSOCIATE de l'ACSE. Le service COSEM-OPEN fournit uniquement la structure qui permet de *transporter* ces informations. L'AP COSEM appropriée est chargée de fournir et de vérifier ces informations.

Sémantique des primitives de service

Les primitives de service COSEM-OPEN doivent fournir des paramètres, comme indiqué dans le Tableau 9.

Tableau 9 – Paramètres de service des primitives de service COSEM-OPEN

	.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 (=)	–	–
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 (=)

	.request	.indication	.response	.confirm
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 (=)	–	–

Les paramètres de service de la primitive de service COSEM-OPEN.request, sauf les paramètres Protocol_Connection_Parameters, User_Information et, en fonction du profil de communications, Service_Class, sont contenus dans les champs de l'APDU AARQ envoyée par le client.

Les paramètres de service de la primitive de service COSEM-OPEN.response, sauf le paramètre Protocol_Connection_Parameters, sont contenus dans les champs de l'APDU AARE envoyée par le serveur.

Le service A-ASSOCIATE et les APDU AARQ et AARE sont indiqués en 7.2. Des exemples de codage sont donnés dans l'Annexe B.

Le paramètre Protocol_Connection_Parameters est obligatoire. Il contient toutes les informations nécessaires pour utiliser les couches du profil de communication, y compris l'identifiant (de protocole) du profil de communication et les adresses requises. Il identifie les participants à l'AA. Les éléments de ce paramètre sont transmis aux entités qui gèrent les connexions aux couches inférieures et aux couches inférieures si nécessaire.

Le paramètre ACSE_Protocol_Version est facultatif. S'il est présent, la valeur par défaut doit être utilisée.

Le paramètre Application_Context_Name est obligatoire. Dans la primitive de demande, il contient la valeur proposée par le client. Dans la primitive de réponse, il contient la même valeur ou celle prise en charge par le serveur.

Le paramètre Calling_AP_Title est conditionnel. Lorsque le paramètre Application_Context_Name indique un contexte d'application utilisant le chiffrement, il doit contenir le titre système du client indiqué en 5.4.8.3.4.6.

L'utilisation des paramètres Called_AP_Title, Called_AE_Qualifier, Called_AP_Invocation_Identifier, Called_AE_Invocation_Identifier, Calling_AE_Qualifier, Calling_AP_Invocation_Identifier et Calling_AE_Invocation_Identifier est facultative. Leur utilisation n'est pas spécifiée dans la présente spécification auxiliaire.

Le paramètre Local_or_Remote est obligatoire. Il indique l'origine de la primitive COSEM-OPEN.confirm. Il est défini sur Remote si la primitive a été générée à la suite de la réception d'une APDU AARE du serveur. Il est défini sur Local si la primitive a été générée localement.

Le paramètre Result est obligatoire. Dans le cas de la confirmation distante, il indique si le serveur a accepté l'AA proposée ou non. Dans le cas de la confirmation locale, il indique si la pile de protocole du côté client a accepté la demande ou non.

Le paramètre Failure_type est obligatoire. Dans le cas de la confirmation distante, il transporte les informations fournies par le serveur. Dans le cas de la confirmation locale et négative, il indique la raison de l'échec.

L'utilisation du paramètre `Responding_AP_Title` est conditionnelle. Lorsque le paramètre `Application_Context_Name` indique un contexte d'application utilisant le chiffrement, il doit contenir le titre système du serveur indiqué en 5.4.8.3.4.6.

L'utilisation des paramètres `Responding_AE_Qualifier`, `Responding_AP_Invocation_Identifier` et `Responding_AE_Invocation_Identifier` est facultative. Leur utilisation n'est pas spécifiée dans la présente spécification auxiliaire.

Le paramètre `ACSE_Requirements` est facultatif. Il permet de sélectionner l'unité fonctionnelle d'authentification facultative du service A-Associate pour l'association. Voir 7.2.1.

La présence du paramètre `ACSE_Requirements` dépend du mécanisme de sécurité utilisé:

- dans le cas du niveau de sécurité le plus faible, il ne doit pas être présent; seule l'unité fonctionnelle Kernel sera utilisée;
- dans le cas du niveau de sécurité faible (LLS), il doit être présent dans la primitive `.request` et peut être présent dans la primitive de service `.response`;
- dans le cas du niveau de sécurité élevé (HLS), il doit être présent dans les primitives de service `.request` et `.response`.

Le paramètre `Security_Mechanism_Name` est conditionnel. Il est présent uniquement si l'unité fonctionnelle d'authentification a été sélectionnée. S'il est présent, la primitive `.request` contient la valeur proposée par le client et la primitive `.response` contient la valeur requise par le serveur, c'est-à-dire celle que le client doit utiliser.

Le paramètre `Calling_Authentication_Value` et les paramètres `Responding_Authentication_Value` sont conditionnels. Ils sont présents uniquement si l'unité fonctionnelle d'authentification a été sélectionnée. Ils contiennent la valeur d'authentification du client/du serveur respectivement, appropriée pour le `Security_Mechanism_Name`.

Le paramètre `Implementation_Information` est facultatif. Son utilisation n'est pas spécifiée dans la présente spécification auxiliaire.

Le paramètre `Proposed_xDLMS_Context` contient les éléments du contexte xDLMS proposé, qui est transporté par l'APDU xDLMS `InitiateRequest`, placé dans le champ des informations sur l'utilisateur de l'APDU `AARQ`.

L'élément `Dedicated_Key` est conditionnel. Il peut être présent uniquement lorsque le paramètre `Application_Context_Name` indique un contexte d'application utilisant le chiffrement. La clé dédiée est utilisée pour le chiffrement dédié des APDU xDLMS échangées dans l'AA établie.

Lorsque la clé dédiée est présente, l'APDU xDLMS `InitiateRequest` doit être chiffrée et authentifiée à l'aide de l'algorithme AES-GCM-128, de la clé de chiffrement unicast globale et de la clé d'authentification (le cas échéant). L'APDU xDLMS `InitiateRequest` doit être chiffrée de la même manière lorsque la clé dédiée est absente, mais il est nécessaire de protéger l'APDU `RLRQ` en incluant le xDLMS `InitiateRequest` chiffré dans son champ d'informations sur l'utilisateur. Voir 6.3.

L'élément `Proposed_DLMS_Version_Number` contient le numéro de version DLMS proposé. Voir 4.2.3.1.

L'élément `Proposed_DLMS_Conformance` contient le bloc de conformité proposé. Voir 7.3.1.

L'élément `Client_Max_Receive_PDU_Size` contient la longueur maximale des APDU xDLMS que le client peut recevoir. Voir le Tableau 1.

Si le contexte xDLMS proposé par le client est acceptable pour le serveur, la primitive de service de réponse doit contenir le paramètre `Negotiated_xDLMS_Context`. Il contient les éléments du contexte xDLMS négocié, contenus dans l'APDU xDLMS `InitiateResponse` et placés dans le champ des informations sur l'utilisateur de l'APDU AARE. Si l'APDU xDLMS `InitiateRequest` a été chiffrée, l'APDU xDLMS `InitiateResponse` doit également être chiffrée de la même manière.

L'élément `Negotiated_DLMS_Version_Number` contient le numéro de version DLMS négocié. Voir 4.2.3.1.

L'élément `Negotiated_DLMS_Conformance` contient le bloc de conformité négocié. Voir 7.3.1.

L'élément `Server_Max_Receive_PDU_Size` contient la longueur maximale des APDU xDLMS que le serveur peut recevoir. Voir le Tableau 1.

L'élément `VAA_name` contient la valeur fictive de 0x0007 dans le cas du référencement LN et l'attribut `base_name` de l'objet Association actuel, 0xFA00, dans le cas du référencement SN.

Si le contexte xDLMS proposé par le client n'est pas acceptable pour le serveur, la primitive de service de réponse doit contenir le paramètre `xDLMS_Initiate_Error`. Il est contenu dans l'APDU `ConfirmedServiceError`, avec les éléments de diagnostic appropriés, et placé dans le champ d'informations sur l'utilisateur de l'APDU AARE.

Le paramètre `User_Information` est facultatif. S'il est présent, il doit être transmis à la couche de support, à condition qu'elle soit capable de le transporter. La primitive d'indication doit alors contenir les informations spécifiques à l'utilisateur, contenues dans la ou les couches de protocole inférieures de support. Voir l'Annexe A.

Le paramètre `Service_Class` est obligatoire. Il indique si le service doit être appelé de manière confirmée ou non confirmée. La gestion de ce paramètre peut dépendre du profil de communication. Voir l'Annexe A.

Utilisation

Un exemple de séquence logique des primitives de service COSEM-OPEN est présenté dans la Figure 8:

- pour l'établissement d'une AA confirmée, avec succès ou non, l'élément a);
- pour l'établissement d'une AA non confirmée, élément b);
- dans le cas d'une AA pré-établie ou d'une tentative infructueuse en raison d'une erreur locale, l'élément c).

La primitive `.request` est appelée par l'AP du client pour demander l'établissement d'une AA confirmée ou non confirmée avec une AP du serveur⁷. À réception de l'appel de demande, l'AL construit et envoie une APDU AARQ au serveur.

La primitive `.indication` est générée par l'AL du serveur à réception d'une APDU AARQ correctement formatée.

La primitive `.response` est appelée par l'AP du serveur pour indiquer à l'AL si l'AA proposée est acceptée ou non. Elle est appelée uniquement si l'AA proposée est confirmée. L'AL construit ensuite une APDU AARE et l'envoie à son homologue, en incluant les paramètres de service reçus de l'AP.

⁷ Avant l'appel de la primitive COSEM-OPEN.request, les couches physiques doivent être connectées. En fonction du profil de communication, l'appel de cette primitive peut également impliquer la connexion d'autres couches inférieures.

La primitive `.confirm` est générée par l'AL du client pour indiquer à l'AP du client si l'AA précédemment demandée est acceptée ou non:

- à distance, à réception d'une APDU AARE;
- localement, si l'AA demandée existe déjà. Cela inclut les AA pré-établies;
- localement, si la primitive `.request` correspondante a été appelée avec `Service_Class == Unconfirmed`;
- localement, si l'AA demandée n'est pas autorisée;
- localement, si une erreur est détectée: paramètres manquants ou incorrects, échecs durant l'établissement des connexions aux couches inférieures demandées, connexion physique manquante, etc.

Le protocole d'établissement d'une AA est indiqué en 7.2.4. Les règles spécifiques au profil de communication sont indiquées à l'Annexe A.

6.3 Service COSEM-RELEASE

NOTE 1 La description a été modifiée pour améliorer l'alignement avec le service A-Release et permettre au champ relatif aux informations sur l'utilisateur de la RLRQ/RLRE de contenir une APDU xDLMS InitiateRequest/InitiateResponse chiffrée afin de sécuriser le service COSEM-RELEASE.

Fonction

La fonction du service COSEM-RELEASE consiste à libérer sans perte de données une AA existante. En fonction de la méthode d'appel de ce service, il utilise le service A-RELEASE de l'ACSE ou non.

Sémantique des primitives de service

Les primitives du service COSEM-RELEASE doivent spécifier des paramètres, comme indiqué dans le Tableau 10.

Tableau 10 – Paramètres de service des primitives de service COSEM-RELEASE

	<code>.request</code>	<code>.indication</code>	<code>.response</code>	<code>.confirm</code>
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(=)

Le paramètre `Use_RLRQ_RLRE` de la primitive `.request` est facultatif. S'il est présent, sa valeur peut être `FALSE` (valeur par défaut) ou `TRUE`. Il indique s'il convient d'utiliser ou non le service A-RELEASE de l'ACSE, qui implique un échange d'APDU RLRQ/RLRE. Le service A-RELEASE et les APDU RLRQ/RLRE sont indiqués en 7.2. Le paramètre `Use_RLRQ_RLRE` de la primitive `.response` est conditionnel. S'il était présent dans la primitive `.indication` et si sa valeur était `TRUE`, il doit également être présent et sa valeur doit être `TRUE`. Sinon, il ne doit pas être présent ou sa valeur doit être `FALSE`.

Si la valeur du paramètre `Use_RLRQ_RLRE` est `FALSE`, l'AA peut être libérée en déconnectant la couche de support de l'AL.

Le paramètre Reason est facultatif. Il peut être présent uniquement si la valeur de Use_RLRQ_RLRE est TRUE. Il est contenu par le champ Reason de l'APDU RLRQ/RLRE respectivement.

Lorsqu'il est utilisé dans la primitive .request, ce paramètre identifie le niveau général d'urgence de la demande. Il est défini sur l'une des valeurs symboliques suivantes:

- normal;
- urgent (non disponible dans DLMS/COSEM), ou
- user defined.

Lorsqu'il est utilisé dans la primitive .response, ce paramètre identifie des informations sur le motif d'acceptation ou de rejet par l'acceptateur de la demande de libération. À noter que dans DLMS/COSEM, le serveur ne peut pas rejeter la demande de libération. Il est défini sur l'une des valeurs symboliques suivantes:

- normal;
- not finished, ou
- user defined.

NOTE 2 La valeur « not finished » est utilisée dans la primitive .response lorsque l'acceptateur est forcé de libérer l'association mais souhaite donner un avertissement indiquant qu'il doit envoyer ou recevoir des informations supplémentaires.

Le paramètre Proposed_xDLMS_Context est conditionnel. Il est présent uniquement si la valeur du paramètre Use_RLRQ_RLRE est TRUE et si l'AA à libérer a été établie avec un contexte d'application utilisant le chiffrement. Cette option permet de sécuriser le service COSEM-RELEASE et d'éviter ainsi une attaque par déni de service qui peut être exécutée en libérant l'AA sans y être autorisé.

Dans la primitive .request, le paramètre Proposed_xDLMS_Context doit être le même que dans la primitive de service COSEM-OPEN.request, une fois l'AA à libérer établie. Il est contenu dans l'APDU xDLMS InitiateRequest, authentifié et chiffré de la même manière que dans l'AARQ et placé dans le champ des informations sur l'utilisateur de l'APDU RLRQ.

Si l'APDU xDLMS InitiateRequest peut être déchiffrée avec succès, la primitive .response doit comporter le même paramètre Negotiated_xDLMS_Context que dans la primitive COSEM-OPEN.response. Il est contenu dans l'APDU xDLMS InitiateResponse, authentifié et chiffré de la même manière que dans l'AARE et placé dans le champ des informations sur l'utilisateur de l'APDU RLRE.

Sinon, l'APDU RLRQ est supprimée de manière silencieuse.

Le paramètre Local_or_Remote est obligatoire. Il indique l'origine de la primitive COSEM-RELEASE.confirm.

Il est défini sur Remote si:

- une APDU RLRE a été reçue du serveur, ou
- une primitive de service de confirmation de la déconnexion a été reçue.

Il est défini sur Local si la primitive a été générée localement.

Le paramètre Result est obligatoire. Dans la primitive .response, il indique si l'AP du serveur peut accepter la demande de libération de l'AA ou non. Étant donné que les serveurs ne peuvent pas refuser de telles demandes, il convient que sa valeur soit normalement SUCCESS, sauf si l'AA référencée n'existe pas.

Le paramètre `Failure_Type` est conditionnel. Il est présent si `Result == ERROR`. Dans ce cas, il indique le motif de l'échec. Il s'agit d'un paramètre généré localement du côté client.

Le paramètre `User_Information` de la primitive `.request` est facultatif. S'il est présent, il est transmis à la couche de support, à condition qu'elle soit capable de le transporter. La primitive `.indication` contient alors les informations spécifiques à l'utilisateur, contenues dans la ou les couches de protocole inférieures de support. De la même manière, il est facultatif dans la primitive `.response`. S'il est présent, il est transmis à la couche de support. Dans la primitive `.confirm`, il peut être présent uniquement lorsque le service est confirmé à distance. Il contient alors les informations spécifiques à l'utilisateur, transportées par la ou les couches de protocole inférieures de support.

La spécification du contenu du paramètre `User_information` n'entre pas dans le domaine d'application de la présente spécification auxiliaire. Voir également l'Annexe A.

Utilisation

Un exemple de séquence logique des primitives de service COSEM-RELEASE est présenté dans la Figure 8:

- pour une libération réussie d'une AA confirmée, élément a);
- pour une libération d'une AA non confirmée, élément b), et
- pour une tentative échouée en raison d'une erreur locale, point c).

L'utilisation du service COSEM-RELEASE dépend de la valeur du paramètre `Use_RLRQ_RLRE`. Lorsqu'il est appelé avec `Use_RLRQ_RLRE == TRUE`, le service est basé sur le service A-RELEASE de l'ACSE. Sinon, l'appel du service entraîne la déconnexion de la couche de support.

La primitive `.request` est appelée par l'AP du client pour demander la libération d'une AA confirmée ou non confirmée avec un AP du serveur. A la réception de l'invocation de la requête avec `Use_RLRQ_RLRE == TRUE`, l'AL construit et envoie une APDU RLRQ au serveur. Sinon, il envoie une primitive `XX-DISCONNECT.request` (où XX désigne la couche de protocole inférieure de support).

La primitive `.indication` est générée par l'AL du serveur si:

- une APDU RLRQ est reçue. Si une erreur de déchiffrement se produit, l'APDU RLRQ est supprimée de manière silencieuse (aucune primitive `.indication` n'est générée), ou
- une primitive `XX-DISCONNECT.request` est reçue.

La primitive `.response` est appelée par l'AP du serveur, mais uniquement si l'AA à libérer est confirmée. À noter que l'AP de serveur ne peut pas refuser cette demande. À réception de la primitive de service `.response`, l'AL du serveur:

- envoie une APDU RLRE si le paramètre `Use_RLRQ_RLRE` est `TRUE`, ou
- envoie une primitive `XX-DISCONNECT.response`.

La primitive `.confirm` est générée par l'AL du client pour indiquer à l'AP du client si la libération demandée de l'AA est acceptée ou non:

- à distance, à réception d'une primitive `XX-DISCONNECT.cnf`. La couche de support est déconnectée, ou
- à distance, à réception d'une APDU RLRE. La couche de support n'est pas déconnectée, ou
- localement, à expiration d'un délai d'attente pour une APDU RLRE, ou
- localement, lors de l'envoi d'une APDU RLRQ pour libérer une AA non confirmée, ou

- localement, lors de la détection d'une erreur locale: paramètres manquants ou incorrects, échec de communication au niveau de la couche de protocole inférieure, etc.

Si l'APDU RLRE reçue contient une APDU xDLMS InitiateResponse chiffrée mais ne peut pas être déchiffrée, l'APDU RLRE doit être supprimée. Le client doit gérer la situation.

Le protocole de libération d'une AA est indiqué en 7.2.5. Les règles spécifiques au profil de communication sont indiquées à l'Annexe A. Voir également 7.2.

6.4 Service COSEM-ABORT

Fonction

La fonction du service COSEM-ABORT est d'indiquer une déconnexion non sollicitée de la couche de support.

Sémantique

Les primitives de service COSEM-ABORT doivent spécifier des paramètres, comme indiqué dans le Tableau 11.

Tableau 11 – Paramètres de service des primitives de service COSEM-ABORT

	.indication
Diagnostics	U

Le paramètre Diagnostics est facultatif. Il doit indiquer le motif possible de la déconnexion et peut également contenir des informations dépendant de la couche de protocole inférieure. La spécification du contenu de ce paramètre n'entre pas dans le domaine d'application de la présente spécification auxiliaire.

Utilisation

La primitive COSEM-ABORT.indication est générée localement du côté client et serveur pour indiquer à l'AP COSEM qu'une connexion à la couche inférieure a été fermée de manière non sollicitée. L'origine d'un tel événement peut être un événement externe (par exemple, la ligne physique est rompue) ou une action d'un AP gestionnaire des connexions à la couche de support, présente dans certains profils, lorsque la couche de support n'est pas gérée par l'AL COSEM. Cela doit entraîner l'annulation par les AP COSEM de toutes les AA existantes, sauf celles pré-établies du côté serveur.

Le protocole du service COSEM-ABORT est détaillé en 7.2.5.3.

6.5 Paramètres de sécurité

Lorsque le chiffrement est utilisé, les services de transfert de données COSEM contiennent des paramètres de sécurité supplémentaires, comme indiqué dans le Tableau 12. Ces paramètres sont communs pour tous les services.

Tableau 12 – Paramètres de sécurité

	.request	.indication	.response	.confirm
Security_Options	C	–	C	–
Security_Status	–	C	–	C
Paramètres de service dépendant du type de service: - GET, SET, ACTION; - Read, Write, UnconfirmedWrite	...	...	...	...
Authentication_Tag	–	C	–	C
NOTE Les primitives de service disponibles dépendent du type de service.				

Le paramètre Security_Options est conditionnel. Il doit contenir les informations suivantes:

- le champ de contrôle de sécurité; voir le Tableau 3;
- si le chiffrement global ou dédié doit être utilisé (ces informations déterminent l'étiquette de l'APDU chiffrée).

Les primitives de service .request et .response doivent utiliser la même suite de sécurité et le même type de chiffrement (global ou dédié).

Le paramètre Security_Status est conditionnel. Il doit contenir les informations suivantes:

- le champ de contrôle de sécurité;
- si le chiffrement global ou dédié a été utilisé (ces informations sont déduites à partir de l'étiquette de l'APDU chiffrée).

Le paramètre Authentication_Tag est conditionnel. Il est présent uniquement si l'APDU a été authentifiée ou authentifiée et chiffrée. Il contient l'étiquette d'authentification calculée à l'aide de la suite de sécurité donnée et des clés.

6.6 Service GET

NOTE La spécification suivante des services utilisant le référencement LN a été modifiée pour rendre la présentation plus cohérente avec celles des services utilisant le référencement SN. Aucune modification technique n'a été introduite.

Fonction

Le service GET est utilisé avec le référencement LN. Il peut être appelé de manière confirmée ou non confirmée. Sa fonction est de lire la valeur d'un ou de plusieurs attributs d'objet d'interface COSEM. Le résultat peut être envoyé dans une seule réponse ou, s'il est trop long, dans plusieurs réponses, avec un transfert de bloc.

Sémantique

Les primitives du service GET doivent fournir des paramètres, comme indiqué dans le Tableau 13.

Tableau 13 – Paramètres de service du service GET

	.request	.indication	.response	.confirm
Invoke_Id	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Service_Class	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 Pour les paramètres de sécurité voir le Tableau 12.				

Le paramètre Invoke_Id identifie l'instance de l'appel de service.

Le paramètre Priority indique le niveau de priorité associé à l'instance de l'appel de service: normale (FALSE) ou haute (TRUE).

Le paramètre Service_Class indique si le service est confirmé ou non confirmé. La gestion de ce paramètre dépend du profil de communication; voir l'Annexe A.

L'utilisation des paramètres Request_Type et Response_Type est décrite dans le Tableau 14.

Tableau 14 – Types de demande et de réponse du service GET

Type de demande		Type de réponse	
NORMAL	La valeur d'un attribut unique est demandée.	NORMAL	Le résultat complet est fourni.
		ONE-BLOCK	Un bloc du résultat est fourni.
NEXT	Le bloc de données suivant est demandé.	ONE-BLOCK	Voir ci-dessus.
		LAST-BLOCK	Le dernier bloc du résultat est fourni.
WITH-LIST	La valeur d'une liste d'attributs est demandée.	WITH-LIST	Le résultat complet est fourni.
		ONE-BLOCK	Voir ci-dessus.
NOTE Le même paramètre Response_Type peut être spécifié plusieurs fois, pour indiquer les réponses possibles pour chaque type de demande.			

Le paramètre COSEM_Attribute_Descriptor se rapporte à un attribut d'objet COSEM. Il est présent lorsque Request_Type == NORMAL ou WITH-LIST. Il s'agit d'un paramètre composite:

- le doublet (COSEM_Class_Id, COSEM_Object_Instance_Id) fait référence de façon non-ambiguë à une seule instance d'objet COSEM;
- l'élément COSEM_Object_Attribute_Id identifie le ou les attributs de l'instance d'objet. COSEM_Object_Attribute_Id == 0 référence tous les attributs publics de l'objet (fonction (attribute_0; voir 4.2.3.10);
- Access_Selection_Parameters figure uniquement lorsque COSEM_Object_Attribute_Id != 0 et si l'accès sélectif de l'attribut donné est disponible; voir 4.2.3.9. Les sous-paramètres Access_Selector et Access_Parameters sont définis dans les définitions d'objet d'interface COSEM; voir la CEI 62056-6-2.

Une primitive de service GET-REQUEST-NORMAL contient un descripteur d'attribut COSEM unique. Une primitive de service GET-REQUEST-WITH-LIST contient une liste de descripteurs d'attributs COSEM; leur nombre est limité par le paramètre server-max-receive-pdu-size: une primitive de service GET-request doit toujours être contenue dans une seule APDU.

Le paramètre Block_Number est utilisé dans la primitive de service GET-REQUEST-NEXT. Il contient le numéro du dernier bloc de données reçu correctement.

Si la forme codée de la réponse passe dans une seule APDU, le paramètre Result est de type Get_Data_Result.

- l'option Data contient la valeur de l'attribut au moment de l'accès, ou
- l'option Data_Access_Result indique la raison de l'échec de lecture de cet attribut.

Une primitive de service GET-RESPONSE-NORMAL contient un seul paramètre Get_Data_Result. Une primitive de service GET-RESPONSE-WITH-LIST contient une liste de paramètres Get_Data_Result; leur nombre et leur ordre doit être identique à celui des paramètres COSEM_Attribute_Descriptor dans la demande.

Si COSEM_Object_Attribute_Id == 0 (Attribute_0), le paramètre Data doit être une structure contenant la valeur de tous les attributs publics dans l'ordre de leur apparition dans la spécification d'objet donnée. Pour les attributs auxquels aucun droit d'accès n'est octroyé dans l'AA donnée ou qui ne sont pas accessibles pour toute autre raison, la valeur NULL doit être renvoyée.

Si la forme codée de la réponse ne peut pas être contenue dans une seule APDU, elle peut être déplacée en blocs de données. L'élément Result est de type DataBlock_G. Il transporte les informations de contrôle de transfert de bloc et l'élément raw-data:

- l'élément Last_Block indique si le bloc actuel est le dernier (TRUE) ou non (FALSE);
- l'élément Block_Number contient le numéro du bloc réel envoyé;
- l'élément Result (interne) transporte le paramètre Raw_Data ou Data_Access_Result. Dans cette formule:
 - si la valeur d'un attribut unique a été demandée, Raw-Data transporte une partie de la valeur de l'attribut (Data). Si la valeur Data ne peut pas être fournie, la réponse doit être de type GET-RESPONSE-NORMAL avec Data_Access_Result;
 - si la valeur d'une liste d'attributs a été demandée, Raw_Data contient une partie de la liste de Get_Data_Results (soit Data, soit Data_Access_Result pour chaque attribut);
 - si les données brutes ne peuvent pas être fournies, Data_Access_Result doit transporter la raison. Pour les codes d'erreur, voir 7.3.3.

Utilisation

Un exemple de séquence logique des primitives de service GET est présenté à la Figure 8:

- pour un GET réussi confirmé, point a);
- pour un GET non confirmé GET point d); et
- pour une tentative échouée en raison d'une erreur locale, point c).

La primitive GET.request est appelée par l'AP du client pour lire la valeur d'un attribut ou de tous les attributs d'un ou de plusieurs objets COSEM de l'AP du serveur. La première demande doit être définie sur le type Request_Type == NORMAL ou WITH-LIST. Une primitive de service GET-REQUEST-NEXT est appelée uniquement lorsque le serveur n'a pas pu livrer l'ensemble des données dans une seule réponse, c'est-à-dire suite à la réception d'une primitive .confirm de type Response_Type == ONE-BLOCK. À réception de la primitive .request, l'AL du client génère l'APDU Get-Request appropriée au type de requête et l'envoie au serveur.

La primitive GET.indication est générée par la couche application du serveur à réception de l'APDU Get-Request.

La primitive GET.response est appelée par l'AP du serveur, si la classe Service_Class == Confirmed, afin d'envoyer une réponse à une primitive .indication reçue. Si l'ensemble des données demandées peut être inséré dans une seule APDU, la primitive .response est appelée avec le type Response_Type == NORMAL ou WITH-LIST, comme il convient. Sinon, elle est appelée avec le type Response_Type == ONE-BLOCK, puis avec LAST-BLOCK.

La primitive GET.confirm est générée par l'AL du client pour accuser réception d'une APDU Get-Response.

Le protocole du service GET est détaillé en 7.3.3.

6.7 Service SET

Fonction

Le service SET est utilisé avec le référencement LN. Il peut être appelé de manière confirmée ou non confirmée. Sa fonction est d'écrire la valeur d'un ou de plusieurs attributs d'objets d'interface COSEM. Les données à écrire peuvent être envoyées dans une seule demande ou, si elle est trop longue pour tenir dans une seule demande, dans plusieurs demandes, grâce au transfert de bloc.

Sémantique

Les primitives de service SET doivent fournir des paramètres, comme indiqué dans le Tableau 15.

Tableau 15 – Paramètres du service SET

	.request	.indication	.response	.confirm
Invoke_Id	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Service_Class	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 Pour les paramètres de sécurité, voir le Tableau 12.				

Le paramètre Invoke_Id identifie l'instance de l'appel de service.

Le paramètre Priority indique le niveau de priorité associé à l'instance de l'appel de service: normale (FALSE) ou haute (TRUE).

Le paramètre Service_Class indique si le service est confirmé ou non confirmé. La gestion de ce paramètre dépend du profil de communication; voir l'Annexe A.

L'utilisation des paramètres Request_Type et Response_Type est décrite dans le Tableau 16.

Tableau 16 – Types de demande et de réponse du service SET

Type de demande		Type de réponse	
NORMAL	La référence d'un attribut unique et l'ensemble des données à écrire sont envoyés.	NORMAL	Le résultat est fourni.
FIRST-BLOCK	La référence d'un attribut unique et le premier bloc de données à écrire sont envoyés.	ACK-BLOCK	La bonne réception du bloc est confirmée.
ONE-BLOCK	L'un des blocs de données à écrire est envoyé.		
LAST-BLOCK	Le dernier bloc de données à écrire est envoyé.	LAST-BLOCK	La bonne réception du bloc est confirmée et le résultat est fourni.
		LAST-BLOCK-WITH-LIST	La bonne réception du dernier bloc est confirmée et la liste des résultats est fournie.
WITH-LIST	La référence d'une liste d'attributs et l'ensemble des données à écrire sont envoyés.	WITH-LIST	La liste des résultats est fournie.
FIRST-BLOCK-WITH-LIST	La référence d'une liste d'attributs et le premier bloc des données à écrire sont envoyés.	ACK-BLOCK	La bonne réception du bloc est confirmée.
NOTE Le même type de réponse (Response_Type) peut figurer plusieurs fois, pour indiquer les réponses possibles pour chaque demande.			

Le paramètre COSEM_Attribute_Descriptor se rapporte à un attribut d'objet COSEM. Il est présent lorsque le type Request_Type == NORMAL, FIRST-BLOCK, WITH-LIST et FIRST-BLOCK-WITH-LIST. Il s'agit d'un paramètre composite:

- le doublet (COSEM_Class_Id, COSEM_Object_Instance_Id) fait référence de façon non-ambiguë à une seule instance d'objet COSEM;
- COSEM_Object_Attribute_Id identifie le(s) attribut(s) de l'instance d'objet. COSEM_Object_Attribute_Id == 0 référence tous les attributs publics de l'objet (fonction (attribute_0; voir 4.2.3.10);
- Access_Selection_Parameters figure uniquement lorsque COSEM_Object_Attribute_Id != 0 et si l'accès sélectif de l'attribut donné est disponible; voir 4.2.3.9. Les sous -paramètres Access_Selector et Access_Parameters sont définis dans les définitions d'objet d'interface COSEM; CEI 62056-6-2.

Une primitive de service SET-REQUEST-NORMAL ou SET-REQUEST-WITH-FIRST-BLOCK contient un seul descripteur d'attribut COSEM. Une primitive de service SET-REQUEST-WITH-LIST ou SET-REQUEST-FIRST-BLOCK-WITH-LIST contient une liste de descripteurs d'attributs COSEM; leur nombre est limité par le paramètre server-max-receive-pdu-size: tous les descripteurs d'attributs COSEM ainsi qu'une partie des données à écrire doivent être contenues dans une seule APDU.

Le paramètre Data contient les données requises pour écrire la valeur des attributs référencés. Le nombre et l'ordre des paramètres Data doivent être identiques à ceux des paramètres COSEM_Attribute_Descriptor.

Si COSEM_Object_Attribute_Id == 0 (Attribute_0), le paramètre Data envoyé doit être une structure des attributs publics respectant leur ordre d'apparition dans la spécification d'objet et contenant pour chacun des attributs une valeur ou la valeur NULL, indiquant que l'attribut n'a pas besoin d'être défini.

Si la forme codée du paramètre Data ne peut pas être contenue dans une seule APDU, elle peut être déplacée en blocs. Le paramètre DataBlock_SA contient des informations de contrôle de transfert de bloc et des données brutes:

- L'élément Last_Block indique si le bloc actuel est le dernier (TRUE) ou non (FALSE);
- L'élément Block_Number contient le numéro du bloc envoyé;
- L'élément Raw_Data contient une partie des données à écrire.

Les paramètres Result figurent dans la primitive .response de type Response_Type != ACK-BLOCK. Leur nombre et leur ordre doivent être identiques à ceux des paramètres COSEM_Attribute_Descriptor de la demande. Chaque paramètre Result doit contenir l'information "réussite" (success) ou la raison de l'échec d'écriture de l'attribut référencé (Data_Access_Result). Dans la primitive .request COSEM_Object_Attribute_Id == 0 (Attribute_0), le paramètre Result doit contenir une liste de résultats, soit l'information "réussite" (success), soit la raison de l'échec d'écriture de l'attribut (Data_Access_Result), pour chaque attribut public, dans l'ordre où ils apparaissent dans la spécification d'objet correspondante.

Le paramètre Block_Number doit être présent lorsque le paramètre Response_Type == ACK-BLOCK, LAST-BLOCK ou LAST-BLOCK-WITH-LIST. Il contient le numéro du dernier bloc de données reçu correctement.

Utilisation

Un exemple de séquence logique des primitives de service SET est présenté à la Figure 8:

- pour un SET réussi confirmé, point a);
- pour un SET non confirmé, point d); et
- pour une tentative échouée en raison d'une erreur locale, point c).

La primitive SET.request est appelée par l'AP du client pour écrire la valeur d'un ou de tous les attributs d'un ou de plusieurs objets COSEM de l'AP du serveur. Si l'ensemble des données à envoyer peuvent être contenues dans une seule APDU, la primitive .request doit être appelée avec le type Request_Type == NORMAL ou WITH-LIST, comme il convient. Sinon, elle doit être appelée avec le type Request_Type == FIRST-BLOCK ou FIRST-BLOCK-WITH-LIST, puis avec le type Request_Type == ONE-BLOCK et enfin avec LAST-BLOCK, comme il convient. A réception de la primitive .request, l'AL du client génère l'APDU Set-Request correspondant au Request_Type et l'envoie au serveur.

La primitive SET.indication est générée par l'AL du serveur à réception de l'APDU Set-Request.

La primitive SET.response est appelée par l'AP du serveur, si Service_Class == Confirmed, pour envoyer une réponse à une primitive .indication reçue. Si les données ont été envoyées dans une seule APDU, la primitive .response est appelée avec Response_Type == NORMAL ou WITH-LIST, comme il convient. Sinon, elle est appelée avec le type Response_Type == ACK-BLOCK, puis avec LAST-BLOCK ou LAST-BLOCK-WITH-LIST, comme il convient.

La primitive SET.confirm est générée par l'AL du client pour accuser réception d'une APDU Set-Response.

Le protocole du service SET est détaillé en 7.3.4.

6.8 Service ACTION

Fonction

Le service ACTION est utilisé avec le référencement LN. Il peut être appelé de manière confirmée ou non confirmée. Sa fonction est d'appeler une ou plusieurs méthodes d'objets d'interface COSEM. Il se décompose en deux phases:

- lors de la première phase, le client envoie la ou les références de la/des méthode(s) à appeler, avec les paramètres d'appel de méthode requis;
- lors de la seconde phase, après l'appel des méthodes, le serveur renvoie le résultat et les paramètres de retour générés par l'appel des méthodes, le cas échéant.

Si les paramètres d'appel de méthode sont trop longs pour être insérés dans une seule requête, ils sont envoyés en plusieurs demandes (transfert de blocs du client vers le serveur). Si les paramètres de résultat et de retour sont trop longs pour être insérés dans une seule réponse, ils sont renvoyés en plusieurs réponses (transfert de blocs du client vers le serveur).

Sémantique

Les primitives du service ACTION doivent fournir des paramètres, comme indiqué dans le Tableau 17.

Tableau 17 – Paramètres du service ACTION

	.request	.indication	.response	.confirm
Invoke_Id	M	M (=)	M (=)	M (=)
Priority	M	M (=)	M (=)	M (=)
Service_Class	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 Pour les paramètres de sécurité, voir le Tableau 12.				

Le paramètre Invoke_Id identifie l'instance de l'appel de service.

Le paramètre Priority indique le niveau de priorité associé à l'instance de l'appel de service: normale (FALSE) ou haute (TRUE).

Le paramètre Service_Class indique si le service est confirmé ou non confirmé. La gestion de ce paramètre dépend du profil de communication; voir l'Annexe A.

L'utilisation des paramètres Request_Type et Response_Type est décrite dans le Tableau 18.

Tableau 18 – Types de demande et de réponse du service ACTION

Type de demande		Type de réponse	
NORMAL	La référence de méthode unique et le paramètre d'appel de méthode complet sont envoyés.	NORMAL	Le résultat et le paramètre de retour complet sont envoyés.
		ONE-BLOCK	Un bloc du résultat et du paramètre de retour sont envoyés.
NEXT	Le bloc de données suivant est demandé.	ONE-BLOCK	Voir ci-dessus.
		LAST-BLOCK	Le dernier bloc du (des) résultat(s) et du (des) paramètre(s) de retour est envoyé.
FIRST-BLOCK	La référence d'une méthode unique et le premier bloc des paramètres d'appel de méthode sont envoyés.	NEXT	La bonne réception du bloc est confirmée.
ONE-BLOCK	Un bloc de paramètres d'appel de méthode est envoyé.		
LAST-BLOCK	Le dernier bloc des paramètres d'appel de méthode est envoyé.	NORMAL	Voir ci-dessus.
		ONE-BLOCK	Voir ci-dessus.
WITH-LIST	La référence d'une liste des méthodes et la liste complète des paramètres d'appel de méthode sont envoyées.	WITH-LIST	La liste complète des résultats et des paramètres de retour est envoyée.
		ONE-BLOCK	Voir ci-dessus.
WITH-LIST-AND-FIRST-BLOCK	La référence d'une liste de méthodes et le premier bloc des paramètres d'appel de méthode sont envoyés.	NEXT	La bonne réception du bloc est confirmée.
NOTE Le même type de réponse (Response_Type) peut figurer plusieurs fois, pour indiquer les réponses possibles pour chaque demande.			

Le paramètre COSEM_Method_Descriptor référence une méthode d'objet COSEM. Il est présent lorsque le type Request_Type == NORMAL, FIRST-BLOCK, WITH-LIST et WITH-LIST-AND-FIRST-BLOCK. Il s'agit d'un paramètre composite:

- le doublet (COSEM_Class_Id, COSEM_Object_Instance_Id) fait référence de façon non-ambiguë à une seule instance d'objet COSEM;
- COSEM_Method_Id identifie une méthode de l'objet COSEM référencé.

Une primitive de service ACTION-REQUEST-NORMAL ou ACTION-REQUEST-FIRST-BLOCK doit contenir un seul descripteur de méthode COSEM. Une primitive de service ACTION-REQUEST-WITH-LIST ou ACTION-REQUEST-WITH-LIST-AND-FIRST-BLOCK doit contenir une liste des descripteurs de méthode COSEM; leur nombre est limité par le paramètre server-max-receive-pdu-size: toutes les références de méthode COSEM – ainsi qu'une partie des paramètres d'invocation de la méthode – doivent être contenues dans une seule APDU.

Le paramètre Method_Invocation_Parameter contient le(s) paramètre(s) nécessaire(s) pour appeler la (les) méthode(s) référencée(s). Si l'invocation d'une méthode ne requiert pas de paramètres supplémentaires, il doit cependant figurer, avec des données NULL.

- si Request_Type == NORMAL, le paramètre Method_Invocation_Parameter est facultatif;
- si Request_Type == WITH-LIST, la primitive de service doit contenir une liste des paramètres d'invocation de la méthode (ou des données NULL). Le nombre et l'ordre des

paramètres d'appel de méthode doivent être identiques à ceux des descripteurs d'attribut COSEM.

Si la forme codée du/des descripteur(s) de méthode COSEM et du/des paramètre(s) d'invocation de méthode ne peut pas être contenue dans une APDU unique, elle peut être transportée en blocs. Le paramètre DataBlock_SA contient des informations de contrôle de transfert de bloc et des données brutes:

- L'élément Last_Block indique si le bloc actuel est le dernier (TRUE) ou non (FALSE);
- L'élément Block_Number contient le numéro du bloc réel envoyé;
- L'élément Raw_Data contient une partie des paramètres d'appel de méthode.

Les paramètres Action_Response sont présents dans la primitive .response lorsque le type Response_Type == NORMAL ou WITH-LIST. Leur nombre et leur ordre doivent être identiques à ceux des descripteurs de méthode COSEM. Ils contiennent deux éléments:

- le paramètre Result. Il contient l'information "réussite" (success) ou la raison de l'échec de l'appel de la méthode référencée (Action-Result);
- Le(s) paramètre(s) Response_Parameter. Chaque paramètre de réponse doit contenir les données retournées suite à l'appel de la méthode, ou une raison de l'échec du retour des paramètres (Data-Access-Result). Si l'appel de l'une des méthodes ne renvoie aucun paramètre, des données NULL doivent être retournées.

Si la réponse ne peut pas être contenue dans une seule APDU, elle peut être transportée en blocs. Le paramètre DataBlock_SA contient des informations de contrôle de transfert de bloc et des données brutes:

- L'élément Last_Block indique si le bloc actuel est le dernier (TRUE) ou non (FALSE);
- L'élément Block_Number contient le numéro du bloc réel envoyé;
- L'élément Raw_Data contient une partie de la réponse:
 - Si une seule méthode est appelée, Raw_Data contient le résultat et les paramètres Response_Parameters le cas échéant. Si aucun paramètre Response_Parameters n'est retourné, la réponse doit être de type ACTION-RESPONSE-NORMAL;
 - Si une liste de méthodes a été appelée, Raw_Data contient une partie de la liste des réponses Action_Responses et des données facultatives.

Le paramètre Block_Number d'une primitive de service ACTION-REQUEST-NEXT doit contenir le numéro du dernier bloc de données reçu correctement en provenance du serveur.

Le paramètre Block_Number inclus dans une primitive de service ACTION-RESPONSE-NEXT doit contenir le le numéro du dernier bloc de données envoyé et reçu correctement en provenance du client.

Utilisation

Un exemple de séquence logique des primitives de service ACTION est présenté à la Figure 8:

- pour une ACTION réussie confirmée, point a);
- pour une ACTION non confirmée point d); et
- pour une tentative échouée en raison d'une erreur locale, point c).

Lors de la première phase, la primitive ACTION.request est appelée par l'AP du client pour appeler une ou plusieurs méthodes d'un ou de plusieurs objets d'interface COSEM de l'AP du serveur. Si la liste complète des descripteurs de méthode et des paramètres d'appel de méthode sont contenus dans une seule APDU, la primitive .request est appelée avec le type Request_Type == NORMAL ou WITH-LIST comme il convient. Sinon, elle est appelée avec le

type Request_Type == FIRST-BLOCK ou WITH-LIST-AND-FIRST-BLOCK, comme il convient, puis avec le type Request_Type == ONE-BLOCK et enfin avec LAST-BLOCK. À réception de la primitive .request, l'AL du client génère l'APDU Action-Request correspondant au type de demande (Request_Type) et l'envoie au serveur.

La primitive ACTION.indication est générée par l'AL du serveur à réception de l'APDU Action-Request.

Lors du transfert de bloc des paramètres d'appel de la méthode, l'AP du serveur appelle la primitive ACTION.response avec le type Request_Type == NEXT jusqu'à réception du dernier bloc.

La primitive ACTION.confirm est générée par l'AL du client à réception de l'APDU Action-Response.

Une fois tous les paramètres d'appel de méthode transférés, le serveur appelle les méthodes des objets d'interface COSEM référencés, puis la deuxième phase commence.

Si la réponse complète est contenue dans une seule APDU, la primitive ACTION.response est appelée par l'AP du serveur avec le type de réponse Response_Type == NORMAL ou WITH-LIST, comme il convient. Sinon, elle est appelée avec le type Response_Type == ONE-BLOCK puis avec LAST-BLOCK. À réception de la primitive .response, l'AL de serveur génère l'APDU Action-Response correspondant au type de réponse (Response_Type) et l'envoie au client.

La primitive ACTION.confirm est générée par l'AL du client pour confirmer la réception de l'APDU Action-Response.

Lors du transfert de bloc des paramètres de retour, l'AP du client appelle la primitive .request avec le type de demande Request_Type == NEXT jusqu'à réception du dernier bloc.

Le protocole du service ACTION est détaillé en 7.3.5.

6.9 Service EventNotification

Fonction

Le service EventNotification est un service de type non-client/serveur non sollicité. Il est demandé par le serveur lorsqu'un événement se produit, afin d'informer le client de la valeur d'un attribut, comme s'il était demandé par le COSEM. Il s'agit d'un service non confirmé.

Sémantique

Les primitives du service EventNotification doivent spécifier des paramètres, comme indiqué dans le Tableau 19.

Tableau 19 – Paramètres de service des primitives du service EventNotification

	.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 (=)

Le paramètre facultatif Time indique l'heure à laquelle la primitive de service EventNotification.request a été émise.

Le paramètre Application_Addresses est facultatif. Il est présent uniquement lorsque le service EventNotification est appelé en dehors d'une AA établie. Dans ce cas, il contient tous les paramètres spécifiques au protocole requis pour identifier les AP de l'expéditeur et du destinataire.

Lorsque la primitive .request ne contient pas le paramètre facultatif Application_Addresses, des adresses par défaut doivent être utilisées. Il s'agit des adresses de l'unité logique de gestion de serveur et de l'AP de gestion du client. Les deux AP sont toujours présentes, et dans tout profil de protocole, elles sont associées à des adresses connues prédéfinies.

Le triplet (COSEM_Class_Id, COSEM_Object_Instance_Id, COSEM_Object_Attribute_Id) référence de façon non-ambiguë un seul attribut de l'instance d'objet d'interface COSEM.

Le paramètre Attribute_Value définit la valeur de cet attribut. Des informations supplémentaires sur l'événement notifié peuvent être obtenues en interrogeant cet objet d'interface COSEM.

Utilisation

Un exemple de séquence logique de primitives de service EventNotification est présenté dans la Figure 8 f) et g).

La primitive .request est appelée par l'AP du serveur pour envoyer la valeur d'un attribut d'objet d'interface COSEM à l'AP de client distant. À réception de la primitive .request, l'AL de serveur génère l'APDU de demande EventNotification.

Dans certains cas, le(s) protocole(s) de couche inférieure de support ne permet(tent) pas l'envoi réel, non sollicité d'une unité de données de protocole. Dans ces cas, le client doit solliciter explicitement l'envoi d'une trame EventNotification, en appelant la primitive de service Trigger_EventNotification_Sending.

La primitive EventNotification.indication est générée par l'AL du client à réception d'une APDU de demande EventNotification.

Le protocole du service EventNotification est détaillé en 7.3.6.

6.10 Service TriggerEventNotificationSending

Fonction

La fonction du service TriggerEventNotificationSending est de déclencher l'envoi de la trame contenant l'APDU EventNotification.request sur le serveur, via le client.

NOTE Ce service est utilisé dans le cadre de protocoles, lorsque le serveur ne peut pas envoyer une APDU EventNotification.request réelle non sollicitée.

Sémantique

La primitive de service TriggerEventNotificationSending.request doit spécifier des paramètres, comme indiqué dans le Tableau 20.

Tableau 20 – Paramètres de service de la primitive de service TriggerEventNotificationSending.request

	.request
Protocol_Parameters	M

Le paramètre Protocol_Parameters contient toutes les informations relatives à la couche de protocole inférieure, requises pour déclencher l'envoi par le serveur d'une trame en attente contenant une APDU EventNotification.request. Ces informations incluent l'identifiant de protocole et tous les paramètres de couches inférieures requis.

Utilisation

À réception d'un appel de service TriggerEventNotificationSending.request envoyé par l'AP du client, l'AL du client doit appeler le service de couche de support correspondant pour qu'il envoie un message déclencheur au serveur.

6.11 Spécification d'accès variable

Variable_Access_Specification est un paramètre des primitives de service .request et .indication suivantes: xDLMS Read / Write / UnconfirmedWrite InformationReport. Ses variantes sont présentées dans le Tableau 21:

- Variable_Name identifie une variable DLMS désignée;
- Le paramètre Parameterized_Access permet d'intégrer des paramètres complémentaires;
- Block_Number_Access contient un numéro de bloc;
- Read_Data_Block_Access transporte les informations de contrôle de transfert de bloc et des données brutes;
- Write_Data_Block_Access transporte les informations de contrôle de transfert de bloc.

L'utilisation des différentes variantes dépend du service utilisé. Elle est détaillée dans les spécifications de service SN correspondantes.

Tableau 21 – Spécification d'accès variable

Variable_Access_Specification	Read .request	Write .request	Write.request non confirmée	Rapport d'informations
Kind_Of_Access	M	M	M	M
Variable_Name	S	S	S	M
Detailed_Access	Non utilisée dans DLMS/COSEM			
Parameterized_Access	S	S	S	–
Variable_Name	M	M	M	
Sélecteur	U	U	U	
Paramètre	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		

6.12 Service Read

Fonction

Le service Read est utilisé avec le référencement SN. Il s'agit d'un service confirmé. Ses fonctions sont les suivantes:

- lire la valeur d'un ou de plusieurs attributs d'objets d'interface COSEM. Dans ce cas, la forme codée de la primitive de service .request doit être contenue dans une APDU unique. Le résultat peut être envoyé dans une seule réponse ou, s'il est trop long, dans plusieurs réponses, avec un transfert de bloc;
- appeler une ou plusieurs méthodes d'objet d'interface COSEM, lorsque des paramètres de retour sont attendus. Dans ce cas, si la primitive de service .request (incluant les références de méthode et les paramètres d'appel de méthode) ou .response (incluant les résultats et les paramètres de retour) est trop longue pour être contenue dans une seule APDU, alors le transfert par blocs, sur plusieurs requêtes et/ou réponses peut être utilisé.

Le service Read est spécifié dans la norme CEI 61334-4-41:1996, 10.4 et Annexe A. Afin de fournir des informations complètes et cohérentes avec la spécification de services à l'aide du référencement LN, la spécification est reproduite ici, avec les extensions adaptées à DLMS/COSEM.

Sémantique

Les primitives du service Read doivent spécifier les paramètres de service tels qu'indiqué dans le Tableau 22.

Tableau 22 – Paramètres du service Read

	.request	.indication	.response	.confirm
Variable_Access_Specification { Variable_Access_Specification }	M	M (=)		
Variable_Name	S	S (=)		
Parameterized_Access	S	S (=)	–	–
Variable_Name	M	M (=)		
Sélecteur	U	U (=)		
Paramètre	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 Pour les paramètres de sécurité, voir le Tableau 12.				

L'utilisation des diverses variantes du paramètre de service Variable-Access-Specification de la primitive de service Read.request et les différents choix de la primitive Read.response sont détaillés dans le Tableau 23.

Tableau 23 – Utilisation des variantes du paramètre Variable_Access_Specification et des choix de Read.response

Read.request Variable_Access_Specification		CHOIX Read.response	
Variable_Name (Variable_Name)	Référence une liste ¹ d'attributs d'objets COSEM.	Data {Data}	Indique la valeur des attributs référencés.
		Data_Access_Error {Data_Access_Error}	Indique la raison de l'échec de lecture.
		Data_Block_Result	Fournit les informations de contrôle de transfert de bloc et un bloc de données brutes.
Parameterized_Access {Parameterized_Access}	Présente une liste ¹ d'attributs d'objets COSEM à lire de façon sélective.	Data {Data}	Voir ci-dessus.
		Data_Access_Error {Data_Access_Error}	
		Data_Block_Result	
	Présente une liste ¹ de méthodes d'objet COSEM, avec les paramètres d'appel de méthode.	Data {Data}	Présente les paramètres de retour d'appel de méthode. NOTE 1 Si les paramètres sont retournés, cela veut dire que l'appel de méthode a réussi.
		Data_Access_Error {Data_Access_Error}	Indique la raison de l'échec de l'appel de méthode.
		Data_Block_Result	Voir ci-dessus.
Read_Data_Block_Access	Contient les informations de contrôle de transfert de bloc et une partie de la forme codée des références de méthode COSEM et des paramètres d'appel de méthode.	Block_Number	Contient le numéro du dernier bloc de données reçu.
Block_Number_Access	Contient le numéro du dernier bloc de données reçu.	Data_Block_Result	Voir ci-dessus.
NOTE 2 Le choix Read.response peut figurer plusieurs fois pour afficher les réponses possibles pour chaque demande.			
¹ Une liste peut contenir plusieurs éléments.			

La primitive de service Read request peut contenir un ou plusieurs paramètres Variable_Access_Specification.

- La variante Variable_Name permet de référencer un attribut d'objet COSEM complet à lire. La demande peut inclure un ou plusieurs noms de variables;
- La variante Parameterized_Access est utilisée soit:
 - pour référencer un attribut d'objet COSEM à lire de façon sélective. Dans ce cas, l'élément Variable_Name référence l'attribut d'objet COSEM, les éléments Sélecteur et Paramètre contiennent respectivement le sélecteur d'accès et les paramètres d'accès comme indiqué dans la spécification de l'attribut; soit
 - pour référencer une méthode d'objet COSEM à appeler. Dans ce cas, l'élément Variable_Name référence la méthode, l'élément Sélecteur est égal à zéro et l'élément Paramètre contient les paramètres d'appel de méthode (le cas échéant) ou des données NULL;
 - la demande peut inclure un ou plusieurs paramètres d'accès paramétré;

NOTE 1 Avec cette spécification, le service Read peut transporter des informations dans les deux directions, tout comme le service ACTION utilisé dans le référencement LN: les paramètres d'appel de méthode du client vers le serveur et les paramètres de retour du serveur vers le client.

- la variante `Read_Data_Block_Access` est utilisée lorsqu'une ou plusieurs méthodes d'objet COSEM sont appelées et lorsque la forme codée de la demande ne rentre pas dans une seule APDU. La demande peut inclure un seul paramètre `Read_Data_Block_Access`. Il transporte les informations de contrôle de transfert de bloc et des données brutes:
 - l'élément `Last_Block` indique si le bloc donné est le dernier (TRUE) ou non (FALSE);
 - l'élément `Block_Number` contient le numéro du bloc réel envoyé;
 - l'élément `Raw_Data` contient une partie de la forme codée de la liste des paramètres `Variable_Access_Specification` (comme en l'absence de transfert de bloc), notamment les références de méthode et les paramètres d'appel de méthode. Ici, seules les variantes `Variable_Name` et `Parameterized_Access` sont autorisées.
- la variante `Block_Number_Access` est appliquée lorsque le serveur utilise le transfert de bloc pour envoyer une réponse longue, pour confirmer la réception d'un bloc de données et pour demander le bloc de données suivant. La demande peut inclure un seul paramètre `Block_Number_Access`. Il contient le numéro du dernier bloc de données reçu correctement.

Le paramètre `Result (+)` indique que le service demandé a réussi.

Sans transfert de bloc, les primitives de service `.response` / `.confirm` contiennent un ou plusieurs paramètres `Read_Result`. Leur nombre et leur ordre doivent être identiques à ceux des paramètres `Variable_Name` / `Parameterized_Access` des primitives `.request` / `.indication`.

Si le service `Read` est utilisé pour lire des attributs, alors:

- l'option `Data` permet d'insérer la valeur de l'attribut au moment de l'accès;
- l'élément `Data_Access_Error` indique la raison de l'échec de lecture de cet attribut.

Si le service `Read` est utilisé pour appeler des méthodes, alors:

- l'option `Data` intègre les paramètres de retour (si des données sont renvoyées, cela signifie que l'appel de méthode a réussi). Si aucun paramètre de retour n'est défini, `Data` ne doit contenir aucune donnée;

NOTE 2 Cependant, si aucune donnée de retour n'est attendue, le service `Write` est utilisé pour appeler des méthodes.

- l'option `Data_Access_Error` indique la raison de l'échec de l'appel de cette méthode.

Dans le cas d'un transfert par bloc, la primitive `.response` / `.confirm` contient un paramètre `Read_Result` unique. L'option `Data_Block_Result` contient un bloc de la réponse:

- l'élément `Last_Block` indique si le bloc donné est le dernier (TRUE) ou non (FALSE);
- l'élément `Block_Number` doit contenir le numéro du bloc envoyé;
- l'élément `Raw_Data` contient une partie de la forme codée de la liste `Read_Results`.

Si le bloc de données ne peut pas être fourni, la primitive `.response` doit contenir un seul paramètre `Result` via le choix `Data_Access_Error` contenant un message d'erreur approprié, par exemple (14) `data-block-unavailable`.

Si le numéro de bloc de la requête n'est pas celui attendu, ou si le bloc suivant ne peut pas être fourni, la primitive de service `Read.response` doit être retournée avec un seul paramètre `Read_Result`, avec le choix `Data_Access_Error`, contenant un code approprié, par exemple (19) `data-block-number-invalid`.

Le choix `Block_Number` est sélectionné lorsque le service `Read` est utilisé pour appeler une ou plusieurs méthodes; la requête est envoyée en plusieurs blocs pour confirmer la bonne

réception du bloc de données et demander le bloc suivant. Il contient le numéro du dernier bloc de données reçu.

Le paramètre Result (–) indique que le service demandé précédemment a échoué. Le paramètre Error_Type indique la raison de l'échec. Dans ce cas, le serveur doit renvoyer une APDU ConfirmedServiceError au lieu d'une APDU ReadResponse.

Utilisation

Un exemple de séquence logique des primitives de service Read est présenté à la Figure 8, point a).

La primitive Read.request est appelée suite à l'appel d'une primitive .request GET ou ACTION par l'AP du client; elle la mappe sur une primitive Read.request via l'ASE SN_MAPPER. L'AL du client génère ensuite l'APDU ReadRequest et l'envoie au serveur. Pour le mappage de service LN / SN, voir 7.3.7.

La primitive Read.indication est générée par l'AL du serveur à réception d'une APDU ReadRequest.

La primitive Read.response est appelée par l'AP du serveur afin d'envoyer une réponse à une primitive Read.indication précédemment reçue. L'AL du serveur génère alors l'APDU ReadResponse et l'envoie au client.

La primitive Read.confirm est générée par l'AL du client après réception d'une APDU ReadResponse. Elle est ensuite mappée vers une primitive .confirm GET ou ACTION par l'ASE SN_MAPPER. La primitive .confirm GET ou ACTION est ensuite générée.

Le protocole du service Read est spécifié en 7.3.7.

6.13 Service Write

Fonction

Le service Write est utilisé avec le référencement SN. Il s'agit d'un service confirmé. Ses fonctions sont les suivantes:

- écrire la valeur d'un ou de plusieurs attributs d'objets d'interface COSEM;
- appeler une ou plusieurs méthodes d'objet d'interface COSEM, lorsqu'aucun paramètre de retour n'est attendu.

Dans les deux cas, si la forme codée de la primitive de service .request ne peut pas être contenue dans une seule APDU, elle peut alors être envoyée en plusieurs requêtes avec transfert par blocs. La primitive de service .response doit toujours pouvoir être contenue dans une seule APDU.

Le service Write est spécifié dans la norme CEI 61334-4-41:1996, 10.5 et Annexe A. Afin de fournir des informations complètes et cohérentes sur la spécification de services avec le référencement LN, la spécification est reproduite ici, avec les extensions adaptées aux caractéristiques DLMS/COSEM.

Sémantique

Les primitives du service Write doivent spécifier les paramètres de service, comme indiqué dans le Tableau 24.

Tableau 24 – Paramètres du service Write

	.request	.indication	.response	.confirm
Variable_Access_Specification { Variable_Access_Specification }	M	M(=)		
Variable_Name	S	S (=)		
Parameterized_Access	S	S (=)	–	–
Variable_Name	M	M (=)		
Sélecteur	M	M (=)		
Paramètre	M	M (=)		
Write_Data_Block_Access	S	S (=)		
Last_Block	M	M (=)	–	–
Block_Number	M	M (=)		
Data { Data }	M	M (=)	–	–
Result (+)	–	–	S	S (=)
Write_Result { Write_Result }	–	–	S	S (=)
Successss			S	S (=)
Data_Access_Error			S	S (=)
Block_Number			S	S (=)
Result (–)			S	S (=)
Error_Type			M	M (=)
NOTE Pour les paramètres de sécurité, voir le Tableau 12.				

L'utilisation des diverses variantes du paramètre de service Variable-Access-Specification de la primitive de service Write request et les différents choix de primitive Write.response sont détaillés dans le Tableau 25. L'utilisation du paramètre de service Data est également expliquée.

**Tableau 25 – Utilisation des variantes de Variable_Access_Specification
et des choix de Write.response**

Write.request Variable_Access_Specification		CHOIX Write.response	
Variable_Name (Variable_Name)	Référence une liste ¹ d'attributs d'objets COSEM.	Success {Success}	Indique que l'attribut référéncé a été écrit avec succès.
	Le paramètre de service Data inclut les données à écrire ou le(s) paramètre(s) d'appel de méthode.	Data_Access_Error {Data_Access_Error}	Indique la raison de l'échec d'écriture.
Parameterized_Access {Parameterized_Access}	Référence une liste ¹ d'attributs d'objet COSEM à écrire de façon sélective.	Success {Success}	Voir ci-dessus.
	Le paramètre de service Data inclut les données à écrire.	Data_Access_Error {Data_Access_Error}	
Write_Data_Block_Access	Contient les informations de contrôle de transfert de bloc. Le paramètre de service Data contient les données brutes, notamment la forme codée de la liste ¹ d'attributs d'objet COSEM ou de références de méthode et la liste des données à écrire ou la liste des paramètres d'appel de méthode.	Block_Number	Contient le numéro du dernier bloc de données reçu.
NOTE Le choix Write.response peut figurer plusieurs fois pour afficher les réponses possibles pour chaque demande.			
¹ Une liste peut contenir plusieurs éléments.			

La primitive de service Write.request peut contenir un ou plusieurs paramètres Variable_Access_Specification:

- la variante Variable_Name permet de référencer un attribut d'objet COSEM complet à écrire ou une méthode d'objet COSEM à appeler. La requête peut inclure un ou plusieurs noms de variables;
- la variante Parameterized_Access est utilisée pour référencer un attribut d'objet COSEM à écrire de façon sélective. Dans ce cas, l'élément Variable_Name référence l'attribut d'objet COSEM, les éléments Sélecteur et Paramètre contiennent respectivement le sélecteur d'accès et les paramètres d'accès comme indiqué dans la spécification de l'attribut. La requête peut inclure un ou plusieurs paramètres Parameterized_Access;

Le paramètre de service Data inclut la ou les valeurs à écrire sur le ou les attributs, ou le(s) paramètre(s) d'appel de la ou des méthodes à appeler. Le nombre et l'ordre des paramètres Data doivent être identiques à ceux des paramètres Variable_Access_Specification.

Si la primitive de service Write.request ne peut pas être contenue dans une seule APDU, le transfert par bloc peut donc être utilisé. Dans ce cas:

- la variante Write_Data_Block_Access du paramètre Variable_Access_Specification contient les informations de contrôle de transfert de blocs:
 - l'élément Last_Block indique si le bloc donné est le dernier (TRUE) ou non (FALSE);
 - l'élément Block_Number contient le numéro du bloc envoyé;
- le paramètre Data contient une partie de la liste des références d'attribut et la liste des données à écrire, ou une partie de la liste de références de méthode et la liste des paramètres d'appel de méthode.
- La requête inclut un seul paramètre Write_Data_Block_Access et un seul paramètre Data.

Le paramètre Result (+) indique que le service demandé a réussi.

Les primitives de service `.response` / `.confirm` contiennent une liste des paramètres `Write_Result`. Leur nombre et leur ordre doivent être identiques à ceux des paramètres `Variable_Name` / `Parameterized_Access` des primitives de service `.request` / `.indication`.

Sans transfert par bloc et avec transfert par bloc après réception du dernier bloc:

- lorsque le service `Write` est utilisé pour écrire un ou plusieurs attributs, chaque élément indique soit que l'accès en écriture a réussi (`Success`), soit la raison de l'échec d'écriture de cette variable (`Data_Access_Error`);
- lorsque le service `Write` est utilisé pour appeler une ou plusieurs méthodes, chaque élément indique soit que l'accès d'appel a réussi (`Success`), soit la raison de l'échec de l'appel de cette variable (`Data_Access_Error`).

Le choix `Block_Number` est utilisé lors du transfert par blocs pour confirmer la réception d'un bloc de données et pour demander le bloc suivant. Il contient le numéro du dernier bloc de données reçu.

Si le numéro de bloc de la requête n'est pas celui attendu, ou si le bloc n'a pas été bien reçu, la primitive de service `Write.response` doit être retournée avec un seul paramètre `Write_Result`, avec le choix `Data_Access_Error`, contenant un code approprié, par exemple (19) `data-block-number-invalid`.

Le paramètre `Result` (–) indique que le service demandé a échoué. Le paramètre `Error_Type` indique la raison de l'échec. Dans ce cas, le serveur doit retourner une `APDU ConfirmedServiceError` au lieu d'une `APDU WriteResponse`.

Utilisation

Un exemple de séquence logique des primitives de service `Write` est présenté à la Figure 8, point a).

La primitive `Write.request` est appelée suite à l'appel d'une primitive `.request SET` ou `ACTION` par l'AP du client; elle la mappe sur une primitive `Write.request` via l'ASE `SN_MAPPER`. L'AL du client génère ensuite l'`APDU WriteRequest` et l'envoie au serveur. Pour le mappage de service `LN / SN`, voir 7.3.8.

La primitive `Write.indication` est générée par l'AL du serveur à réception d'une `APDU WriteRequest`.

La primitive `Write.response` est appelée par l'AP du serveur afin d'envoyer une réponse à une primitive `Write.indication` reçue précédemment. L'AL du serveur génère alors l'`APDU WriteResponse` et l'envoie au client.

La primitive `Write.confirm` est générée par l'AL du client après réception d'une `APDU WriteResponse`. Elle est ensuite mappée vers une primitive `.confirm SET` ou `ACTION` par l'ASE `SN_MAPPER`. La primitive `.confirm SET` ou `ACTION` est ensuite générée.

Le protocole du service `Write` est détaillé en 7.3.8.

6.14 Service UnconfirmedWrite

Fonction

Le service `UnconfirmedWrite` est utilisé avec le référencement `SN`. Il s'agit d'un service non confirmé. Ses fonctions sont les suivantes:

- écrire la valeur d'un ou plusieurs attributs d'objets `COSEM`;

- appeler une ou plusieurs méthodes d'objet d'interface COSEM, lorsqu'aucun paramètre de retour n'est attendu.

La primitive de service UnconfirmedWrite.request doit toujours pouvoir être contenue dans une seule APDU.

Le service UnconfirmedWrite est spécifié dans la norme CEI 61334-4-41:1996, 10.6 et Annexe A. Afin de fournir des informations complètes et cohérentes sur la spécification de services avec le référencement LN, la spécification est reproduite ici, avec les extensions adaptées aux caractéristiques DLMS/COSEM.

Sémantique

Les primitives du service UnconfirmedWrite doivent spécifier les paramètres de service, comme indiqué dans le Tableau 26.

Tableau 26 – Paramètres du service UnconfirmedWrite

	request	Indication
Variable_Access_Specification { Variable_Access_Specification }	M	M(=)
Variable_Name	S	S(=)
Parameterized_Access	S	S(=)
Variable_Name	M	M(=)
Sélecteur	M	M(=)
Paramètre	M	M(=)
Data { Data }	M	M(=)
NOTE Pour les paramètres de sécurité, voir le Tableau 12.		

L'utilisation des diverses variantes du paramètre de service Variable-Access-Specification de la primitive de service UnconfirmedWrite.request est détaillée dans le Tableau 27. L'utilisation du paramètre de service Data est également expliquée.

Tableau 27 – Utilisation des variantes de Variable_Access_Specification

UnconfirmedWrite.request Variable_Access_Specification	
Variable_Name {Variable_Name}	Référence un attribut d'objet COSEM. Le paramètre de service Data inclut les données à écrire ou le(s) paramètre(s) d'appel de méthode.
Parameterized_Access {Parameterized_Access}	Référence un attribut d'objet COSEM avec accès sélectif. Le paramètre de service Data inclut les données à écrire.

La primitive de service UnconfirmedWrite.request peut contenir un ou plusieurs paramètres Variable_Access_Specification:

- la variante Variable_Name permet de référencer un attribut d'objet COSEM complet à écrire ou une méthode d'objet COSEM à appeler;
- la variante Parameterized_Access est utilisée pour référencer un attribut d'objet COSEM à écrire de façon sélective. Dans ce cas, l'élément Variable_Name référence l'attribut d'objet COSEM, les éléments Sélecteur et Paramètre contiennent respectivement le sélecteur d'accès et les paramètres d'accès comme indiqué dans la spécification de l'attribut.

Le paramètre de service Data inclut la ou les valeurs à écrire sur le ou les attributs, ou le(s) paramètre(s) d'appel de la ou des méthodes à appeler. Le nombre et l'ordre des paramètres Data doivent être identiques à ceux des paramètres Variable_Access_Specification.

Utilisation

Un exemple de séquence logique des primitives de service Write est présenté à la Figure 8, point d).

La primitive UnconfirmedWrite.request est appelée suite à l'appel d'une primitive .request SET ou ACTION avec le paramètre Service_Class == Unconfirmed par l'AP du client; elle la mappe vers une primitive UnconfirmedWrite.request via l'ASE SN_MAPPER. L'AL du client génère ensuite l'APDU UnconfirmedWriteRequest et l'envoie au serveur.

La primitive UnconfirmedWrite.indication est générée par l'AL du serveur à réception d'une APDU WriteRequest.

Le protocole du service UnconfirmedWrite est spécifié en 7.3.9.

6.15 Service InformationReport

Fonction

Le service InformationReport est un service de type non-client/serveur non sollicité. Il est demandé par un serveur utilisant le référencement SN lorsqu'un événement se produit, afin d'informer le client de la valeur d'une ou plusieurs variables DLMS désignées, mappées vers des attributs d'objet d'interface COSEM, comme si les valeurs avaient été demandées par le client. Il s'agit d'un service non confirmé.

Le service InformationReport est détaillé dans la norme CEI 61334-4-41:1996, 10.7 et Annexe A. Afin de fournir des informations complètes et cohérentes sur la spécification de services à l'aide du référencement LN, la spécification du service InformationReport est reproduite ici, avec les extensions adaptées aux caractéristiques DLMS/COSEM.

Sémantique

Les primitives du service InformationReport doivent spécifier des paramètres, comme indiqué dans le Tableau 28.

Tableau 28 – Paramètres du service InformationReport

	.request	.indication
Current_Time	M	M (=)
Variable_Access_Specification { Variable_Access_Specification }	M	M (=)
Variable_Name	M	M (=)
Data { Data }	M	M(=)

Le paramètre Current_Time indique l'heure à laquelle la primitive de service InformationReport.request a été émise.

Le paramètre Variable_Access_Specification du choix Variable_Name indique une ou plusieurs variables DLMS désignées – mappées sur des attributs d'objet d'interface COSEM – dont les valeurs sont envoyées par le serveur.

Le paramètre Data contient la valeur des variables DLMS désignées, qui respectent l'ordre des paramètres Variable_Access_Specification.

Le protocole du service InformationReport est détaillé en 7.3.10.

6.16 Services de gestion de couches côté client: Demande SetMapperTable.request

Fonction

La fonction du service SetMapperTable est de gérer l'ASE SN_MAPPER du client.

Sémantique

Une seule primitive est requise: .request. Elle doit spécifier les paramètres suivants (voir Tableau 29):

Tableau 29 – Paramètres de service des primitives de service SetMapperTable.request

	request
Mapping_Table	M

Le paramètre Mapping_table est obligatoire. Il contient les éléments de l'attribut « object_list » du serveur et de l'AA demandés. La structure du contenu est définie dans la CEI 62056-6-2.

Utilisation

Le service SetMapperTable.request est appelé par l'AP du client afin de fournir des informations de mappage vers l'ASE du client SN_MAPPER. Ce service ne déclenche pas de transmission de données entre le client et le serveur. L'AP du client utilise cette primitive de service, afin d'améliorer l'efficacité du processus de mappage si le référencement SN est utilisé.

6.17 Récapitulatif des services et du mappage de service de transfert de données LN/SN

Les Tableaux 30, 31 et 32 suivants récapitulent les services de couche application COSEM.

Tableau 30 – Récapitulatif des services ACSE

Côté client	Côté serveur
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

Tableau 31 – Récapitulatif des services xDLMS pour le référencement LN

Côté client	Côté serveur
GET.request	GET.indication
GET.confirm	GET.response
SET.request	SET.indication
SET.confirm	SET.response
ACTION.request	ACTION.indication
ACTION.confirm	ACTION.response
EventNotification.indication	EventNotification.request
Trigger_EventNotification_Sending.request	–

Tableau 32 – Récapitulatif des services xDLMS pour le référencement SN

Côté client	Côté serveur
Read.request	Read.indication
Read.confirm	Read.response
Write.request	Write.indication
Write.confirm	Write.response
UnconfirmedWrite.request	UnconfirmedWrite.indication
InformationReport.indication	InformationReport.request

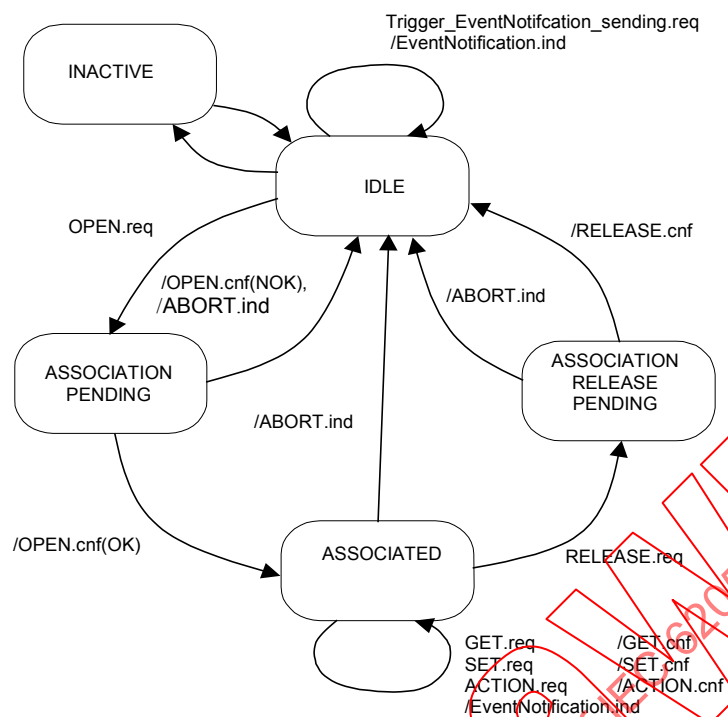
Lorsque le serveur utilise le référencement SN, un mappage entre les primitives de service utilisant le référencement LN et le référencement SN s'effectue côté client. Ce mappage est détaillé en 7.3.7, 7.3.8, 7.3.9 et 7.3.10.

7 Spécification du protocole de couche application COSEM

7.1 Fonction de commande

7.1.1 Définitions des états de la fonction de commande côté client

La Figure 9 montre la machine à état de la CF côté client; voir également la Figure 1.



IEC 1120/13

NOTE Sur les schémas d'état des CF côté client et serveur, les conventions suivantes sont utilisées:

- les primitives de service sans caractère initial "/" sont des "stimulants": l'appel de ces primitives constitue l'origine du changement d'état;
- les primitives de service avec un caractère initial "/" sont des "sorties": la génération de ces primitives s'effectue au moment du changement d'état.

Figure 9 –Machine à état, partielle, pour la fonction de commande côté client

Les définitions d'état de la CF côté client (et de l'AL où figure la CF) sont les suivantes:

INACTIVE Dans cet état, la CF ne présente aucune activité: elle ne fournit aucun service à l'AP et elle n'utilise aucun service de la couche de protocole de support.

IDLE Dans cet état, la CF ne présente aucune AA existante en cours de libéralisation ou d'établissement⁸. Néanmoins, certains échanges de données entre le client et le serveur (si le canal physique a été établi précédemment) sont possibles. Dans cet état, la CF peut gérer le service EventNotification.

NOTE Les transitions entre les états INACTIVE et IDLE sont contrôlées en dehors du protocole. Par exemple, on peut considérer que la CF effectue la transition de l'état INACTIVE à l'état IDLE en étant instanciée et placée en haut de la couche de protocole de support. La transition inverse s'effectue en supprimant l'instance donnée de la CF.

ASSOCIATION PENDING La CF quitte l'état IDLE et passe à cet état lorsque l'AP demande l'établissement d'une AA en appelant la primitive COSEM-OPEN.request (OPEN.req). La CF peut quitter cet état et passer soit à l'état ASSOCIATED, soit à l'état IDLE. Elle génère la primitive COSEM-OPEN.confirm, (/OPEN.cnf(OK)) ou (/OPEN.cnf(NOK)), en fonction du résultat de la demande d'association. La CF quitte également cet état et revient à l'état IDLE lorsqu'elle génère la primitive COSEM-ABORT.indication (/ABORT.ind).

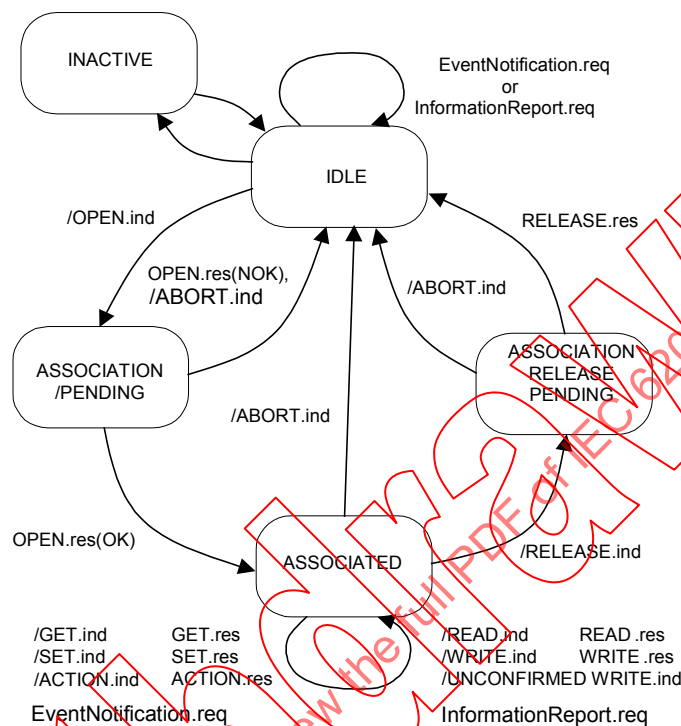
ASSOCIATED La CF passe à cet état lorsque l'AA a été établie avec succès. Les services de données de type client/serveur sont disponibles dans cet état, et le service EventNotification peut être utilisé. La CF reste à cet état jusqu'à ce que l'AP demande la libération de l'AA en appelant la primitive COSEM-RELEASE.request (RELEASE.req). La CF quitte également cet état et revient à l'état IDLE lorsqu'elle génère la primitive COSEM-ABORT.indication (/ABORT.ind).

ASSOCIATION RELEASE La CF quitte l'état ASSOCIATED et passe à cet état lorsque l'AP demande la libération de l'AA en appelant la primitive COSEM-RELEASE.request (RELEASE.req). La CF reste sous cet état et attend que le serveur réponde à cette demande. Le serveur n'est pas autorisé à refuser une

⁸ Noter qu'il s'agit de l'état de la machine pour l'AL: les connexions de couche inférieure, notamment la connexion physique, ne sont pas prises en compte. D'un autre côté, l'établissement de la connexion physique s'effectue en dehors du protocole.

PENDING demande de libération. Après avoir quitté cet état, la CF passe donc toujours à l'état IDLE. La CF peut quitter cet état en générant la primitive COSEM-RELEASE.confirm après réception de la réponse envoyée par le serveur ou en la générant localement (/RELEASE.cnf). La CF quitte également cet état et revient à l'état IDLE lorsqu'elle génère la primitive COSEM-ABORT.indication (/ABORT.ind).

7.1.2 Définitions des états de la fonction de commande côté serveur



IEC 1121/13

Légende

Anglais	Français
Event Notification.req or InformationReport.req	Event Notification.req ou InformationReport.req

Figure 10 – État partiel de la machine pour la fonction de commande côté serveur

La Figure 10 montre l'état de la machine de la CF côté serveur; voir également la Figure 1.

INACTIVE Dans cet état, la CF ne présente aucune activité: elle ne fournit aucun service à l'AP et elle n'utilise aucun service de la couche de protocole de support.

IDLE Dans cet état, la CF ne présente aucune AA existante, ni en cours de libération ni en cours d'établissement⁹. Néanmoins, certains échanges de données entre le client et le serveur sont possibles si le canal physique a été établi précédemment. La CF peut gérer les services EventNotification / InformationReport.

ASSOCIATION PENDING La CF quitte l'état IDLE et passe à cet état lorsque le client demande l'établissement d'une AA et lorsque l'AL du serveur génère la primitive COSEM-OPEN.indication (/OPEN.ind). La CF peut quitter cet état et passer soit à l'état ASSOCIATED, soit à l'état IDLE, en fonction du résultat de la demande d'association, et appeler la primitive COSEM-OPEN.response, (/OPEN.res(OK)) ou (/OPEN.res(NOK)). La CF quitte également cet état et revient à l'état IDLE lorsqu'elle génère la primitive COSEM-ABORT.indication (/ABORT.ind).

ASSOCIATED La CF passe à cet état lorsque l'AA a été établie avec succès. Les services de données de type

⁹ Noter qu'il s'agit de l'état de la machine pour l'AL: les connexions de couche inférieure, notamment la connexion physique, ne sont pas prises en compte. D'un autre côté, l'établissement de la connexion physique s'effectue en dehors du protocole.

client/serveur sont disponibles dans cet état, et les services EventNotification / InformationReport peuvent être utilisés. La CF reste dans cet état jusqu'à ce que le client demande la libération de l'AA et que l'AL du serveur génère la primitive COSEM-RELEASE.ind (/RELEASE.ind). La CF quitte également cet état et revient à l'état IDLE lorsqu'elle génère la primitive COSEM-ABORT.indication (/ABORT.ind).

ASSOCIATION La CF quitte l'état ASSOCIATED et passe dans cet état lorsque le client demande la libération de l'AA et l'AP du serveur reçoit la primitive COSEM-RELEASE.indication (/RELEASE.ind). La CF reste dans cet état, en attendant que l'AP accepte la demande de libération. Le serveur n'est pas autorisé à refuser une demande de libération. Après avoir quitté cet état, la CF passe donc toujours à l'état IDLE. La CF peut quitter cet état lorsque l'AP accepte la libération de l'AA, puis appelle la primitive COSEM-RELEASE.response (RELEASE.res). La CF quitte également cet état et revient à l'état IDLE lorsqu'elle génère la primitive COSEM-ABORT.indication (/ABORT.ind).

RELEASE

PENDING

7.2 Services ACSE et APDU

7.2.1 Unités fonctionnelles ACSE, services et paramètres de service

L'ACSE d'AL COSEM est basé sur l'ACSE orienté connexion, comme indiqué dans les normes ISO/CEI 15953 et ISO/CEI 15954.

Les unités fonctionnelles permettent de négocier les exigences d'utilisateur ACSE lors de l'établissement de l'association. Trois unités fonctionnelles sont définies:

- unité fonctionnelle Kernel;
- unité fonctionnelle d'authentification, et
- unité fonctionnelle Application Context Negotiation.

L'AL COSEM utilise uniquement les unités fonctionnelles Kernel et Authentication.

Les paramètres d'exigences ACSE des APDU AARQ et AARE permettent de sélectionner les unités fonctionnelles de l'association.

L'unité fonctionnelle Kernel est toujours disponible. Il s'agit de l'unité fonctionnelle par défaut. Pour être incluse, l'unité fonctionnelle d'authentification doit être explicitement demandée sur l'APDU AARQ et être acceptée sur l'APDU AARE.

Lorsque l'unité fonctionnelle d'authentification est sélectionnée, les paramètres complémentaires des APDU AARQ et AARE sont pris en charge. L'unité fonctionnelle d'authentification n'affecte pas les éléments de la procédure. Le Tableau 33 affiche les services et les paramètres associés aux unités fonctionnelles ACSE, utilisés par l'AL COSEM. La syntaxe abstraite des APDU ACSE sont spécifiés à l'Article 8.

Tableau 33 – Unités fonctionnelles ACSE, services et paramètres de service

Unité fonctionnelle	Service	APDU	paramètre	Présence
Kernel	A-ASSOCIATE	AARQ	protocol-version	O ¹⁾
			application-context-name	M
			called-AP-title	U
			called-AE-qualifier	U
			called-AP-invocation-identifiant	U
			called-AE-invocation-identifiant	U
			calling-AP-title	U
			calling-AE-qualifier	U
			calling-AP-invocation-identifiant	U
			calling-AE-invocation-identifiant	U
			implementation-information	O
			user-information ²⁾	M
			(contenant une APDU xDLMS-Initiate.request)	
			dedicated-key	U
			response-allowed	U
			proposed-quality-of-service	U
			proposed-dlms-version-number	M
			proposed-conformance	M
			client-max-receive-pdu-size	M
		AARE	protocol-version	O
			application-context-name	M
			result	M
			result-source-diagnostic	M
			responding-AP-title	U
			responding-AE-qualifier	U
			responding-AP-invocation-identifiant	U
			responding-AE-invocation-identifiant	U
			implementation-information	O
			user-information ²⁾	M
			(contenant une APDU xDLMS-Initiate.request)	S
			negotiated-quality-of-service	U
			negotiated-dlms-version-number	M
			negotiated-conformance	M
			server-max-receive-pdu-size	M
			vaa-name	M
			(ou contenant une APDU ConfirmedServiceError)	S
	A-RELEASE	RLRQ	reason	U
			user-information	U
		RLRE	reason	U
			user-information	U

Unité fonctionnelle	Service	APDU	paramètre	Présence
Authentification	A-ASSOCIATE	AARQ	sender-acse-requirements	U
			mechanism-name	U
			calling-authentication-value	U
		AARE	responder-acse-requirements	U
			mechanism-name	U
			responding-authentication-value	U

¹⁾ O: La présence est une option ACPM (Couche application COSEM).

²⁾ Selon la norme ISO/CEI 15953, le paramètre user-information est facultatif. Cependant, dans l'environnement COSEM, il est obligatoire dans les APDU AARQ / AARE.

En général, la valeur de chaque paramètre de l'APDU AARQ est déterminée par les paramètres de la primitive de service COSEM-OPEN.request. De la même façon, la valeur de chaque paramètre de l'AARE est déterminée par la primitive COSEM-OPEN.response. Le service COSEM-OPEN est spécifié en 6.2.

Les paramètres des APDU AARQ et AARE sont spécifiés ci-dessous. La gestion de ces paramètres est détaillée en 7.2.4.1.

- **protocol-version**: l'AL COSEM utilise la valeur par défaut. Pour plus de détails, voir la norme ISO/CEI 15954;
- **application-context-name**: Les noms contextuels d'application COSEM sont indiqués en 7.2.2.2;
- **called-, calling- et responding- titles, qualifieurs et invocation-identifieurs**: ces champs facultatifs contiennent la valeur des paramètres correspondants du service COSEM-OPEN. Pour plus de détails, voir la norme ISO/CEI 15954;
- **implementation-information**: ce paramètre n'est pas utilisé par l'AL COSEM. Pour plus de détails, voir la norme ISO/CEI 15954;
- **user-information**: l'APDU AARQ contient une APDU xDLMS InitiateRequest où sont définis les éléments du paramètre Proposed_xDLMS_Context de la primitive de service COSEM-OPEN.request. L'APDU AARE contient une APDU xDLMS InitiateResponse où figurent les éléments du paramètre Negotiated_xDLMS_Context ou une APDU xDLMS ConfirmedServiceError où figurent les éléments du paramètre xDLMS_Initiate_Error de la primitive de service COSEM-OPEN.response.
- **sender- et responder-acse-requirements**: ce paramètre permet de sélectionner les unités fonctionnelles facultatives des APDU AARQ et AARE. Dans COSEM, seule l'unité fonctionnelle d'authentification est utilisée. Lorsqu'elle est présente, elle contient la valeur BIT STRING { authentication (0) }. Bit positionné: module d'authentification sélectionné;
- **mechanism-name**: Les noms des mécanismes d'authentification COSEM sont indiqués en 7.2.2.3;
- **calling- et responding- authentication-value**: voir 5.3;
- **result**: la valeur de ce paramètre est déterminée par l'AP COSEM (acceptor) ou l'AL COSEM (ACPM) comme indiqué ci-dessous. Elle est utilisée pour déterminer la valeur du paramètre Result de la primitive COSEM-OPEN.confirm:
 - Si l'APDU AARQ est rejetée par l'ACPM (autrement dit, la primitive COSEM-OPEN.indication n'est pas émise par l'AL COSEM), la valeur "rejected (permanent)" ou "rejected (transient)" est attribuée par l'ACPM;
 - sinon, la valeur est déterminée par le paramètre Result de l'APDU COSEM-OPEN.response;

- **result-source-diagnostic**: ce paramètre contient la valeur Result source et la valeur Diagnostic. Il est utilisé pour déterminer la valeur du paramètre Failure_Type de la primitive COSEM-OPEN.confirm:
 - Valeur Result source: si l'AARQ est rejetée par l'ACPM, (autrement dit, la primitive COSEM-OPEN.indication n'est pas émise par l'AL COSEM), l'ACPM attribue la valeur "ACSE service-provider". Sinon, l'ACPM attribue la valeur "ACSE service-user";
 - Valeur Diagnostic: Si l'AARQ est rejetée par l'ACPM, la valeur appropriée est attribuée par l'ACPM. Sinon, la valeur est déterminée par le paramètre Failure_Type de la primitive COSEM-OPEN.response. Si le paramètre Diagnostic ne figure pas dans la primitive .response, l'ACPM attribue la valeur "null".

Les paramètres des APDU RLRQ / RLRE - utilisés lorsque le service COSEM-RELEASE (voir 6.3) est appelé avec le paramètre Use_RLRQ_RLRE == TRUE - sont spécifiés ci-dessous.

- **reason**: indique la valeur adéquate, comme indiqué en 6.3;
- **user-information**: si ces informations sont présentes, elles contiennent une APDU xDLMS InitiateRequest / InitiateResponse, où figurent les éléments du paramètre Proposed_xDLMS_Context / Negotiated_xDLMS_Context, respectivement des primitives de service COSEM-RELEASE.request / .response; voir 6.3.

7.2.2 Noms COSEM enregistrés

7.2.2.1 Généralités

Dans un environnement OSI, de nombreux types d'objets réseau doivent être identifiés par des noms non-ambigus au niveau mondial. Ces objets réseau sont des syntaxes abstraites, des syntaxes de transfert, des contextes d'application, des noms de mécanismes d'authentification, etc. Généralement, les noms de ces objets sont attribués par le comité qui développe la norme de base ISO particulière ou durant des ateliers pour implémenteurs. Il convient de déposer ces noms d'objets. Pour DLMS/COSEM, ces noms d'objets sont attribués par l'association DLMS User Association. Ils sont spécifiés ci-dessous.

La décision no. 1999.01846 de l'OFCOM, Suisse, attribue le préfixe suivant aux identifiants d'objet définis par l'association DLMS User Association.

{ joint-iso-ccitt(2) country(16) country-name(756) identified-organisation(5) DLMS-UA(8) }

Pour DLMS/COSEM, les identifiants d'objet sont spécifiés afin de désigner les éléments suivants:

- Noms de contexte d'application COSEM;
- Noms de mécanismes d'authentification COSEM.

7.2.2.2 Contexte d'application COSEM

Pour établir un échange d'informations efficace à l'intérieur de l'AA, la paire d'appels-AE doit être mutuellement au courant et respecter un ensemble de règles relatives à cet échange. Cet ensemble de règles communes est appelé le contexte d'application de l'AA. Le contexte d'application relatif à une AA est déterminé lors de son établissement¹⁰. Les méthodes possibles sont les suivantes:

- définition d'un contexte d'application pré-existant;
- transfert d'une description réelle du contexte d'application.

¹⁰ Un seul contexte d'application est défini par AA. Cependant, l'ensemble de règles constitutives du contexte d'application d'une AA peut contenir des règles modifiant cet ensemble pour la durée de l'AA.

Dans un environnement COSEM, un contexte d'application pré-existe. Il est référencé sous son nom lors de l'établissement de l'AA. Le nom de contexte d'application est défini comme suit: IDENTIFIANT OBJET ASN.1. COSEM identifie le nom de contexte d'application en fonction de la valeur d'identifiant d'objet suivante:

COSEM_Application_Context_Name::=

{joint-iso-ccitt(2) country(16) country-name(756) identified-organisation(5) DLMS-UA(8) application-context(1) context_id(x)}

La signification de ces contextes d'application COSEM généraux est la suivante:

- deux ASE sont présents dans l'appel AE, l'ACSE et le xDLMS_ASE;
- le xDLMS_ASE doit être tel que défini dans la CEI 61334-4-41¹¹;
- la syntaxe de transfert doit être de type A-XDR, spécifié dans la CEI 61334-6.

Les ID de contexte spécifiques et l'utilisation des APDU chiffrées et non-chiffrées sont indiqués dans le Tableau 34 ci-dessous:

Tableau 34 – Utilisation des APDU chiffrées et non chiffrées

Application_Context_Name	context_id	APDU non chiffrées	APDU chiffrées
Logical_Name_Referencing_No_Ciphering::=	context_id(1)	Oui	Non
Short_Name_Referencing_No_Ciphering::=	context_id(2)	Oui	Non
Logical_Name_Referencing_With_Ciphering::=	context_id(3)	Oui	Oui
Short_Name_Referencing_With_Ciphering::=	context_id(4)	Oui	Oui

Pour établir avec succès une AA, il convient que le paramètre application-context-name des APDU AARQ et AARE contienne l'un des noms « valides ». Le client propose un nom de contexte d'application via le paramètre Application_Context_Name de la primitive COSEM-OPEN.request. Le serveur peut retourner toute valeur, soit la valeur proposée, soit la valeur qu'il prend en charge.

7.2.2.3 Nom de mécanisme d'authentification COSEM

L'authentification du client, du serveur ou les deux sont l'un des aspects de sécurité traités par la spécification DLMS/COSEM. Trois niveaux de sécurité d'authentification sont spécifiés:

- pas de sécurité d'authentification (niveau inférieur);
- niveau de sécurité d'authentification bas basé sur un mot de passe (LLS) identifiant le client uniquement;
- niveau de sécurité d'authentification 4 passes de haut niveau (HLS) identifiant aussi bien le client et le serveur.

La COSEM identifie les mécanismes d'authentification en fonction de la valeur d'identifiant d'objet générale suivante:

¹¹ Avec les extensions COSEM sur les DLMS; voir 4.2.3.1.

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)}

La valeur de l'élément mechanism_id sélectionne l'un des mécanismes de sécurité spécifiés:

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)
NOTE Avec l'élément mechanism_id(2), la méthode de traitement de la demande d'accès est secrète.	

Lorsque l'élément Authentication_Mechanism_Name est présent dans le service COSEM-OPEN, l'unité fonctionnelle d'authentification du service A-ASSOCIATE doit être sélectionnée. Le processus d'authentification LLS et HLS est décrit en 5.3 et 5.4.8.4.

7.2.3 Règles de codage d'APDU

Les APDU ACSE doivent être codées en BER, conformément à l'ISO/CEI 8825-1. Le paramètre user-information de ces APDU doit contenir l'APDU xDLMS InitiateRequest / InitiateResponse / ConfirmedServiceError appropriée, codée en A-XDR. L'OCTET STRING résultant doit être codé en BER.

Des exemples de codage d'APDU AARQ/AARE sont fournis à l'Annexe B et à l'Annexe C.

7.2.4 Protocole d'établissement d'association d'applications

7.2.4.1 Protocole d'établissement d'associations d'applications confirmées

L'établissement d'AA à l'aide du service A-Associate de l'ACSE est l'élément clé de l'interopérabilité COSEM. Les éléments d'une AA sont:

- l'AP du client, qui propose les AA, et
- l'AP du serveur¹², qui les accepte ou non.

La Figure 11 indique le MSC lorsque:

- la primitive COSEM-OPEN.request demande une AA confirmée;
- la connexion de couches inférieures de support est requise pour l'établissement de cette AA.

Un AP de client souhaitant établir une AA confirmée appelle la primitive COSEM-OPEN.request de l'ASO avec le paramètre Service_Class == Confirmed. Le paramètre response-allowed de l'APDU xDLMS InitiateRequest est défini sur TRUE. L'AL de client attend une APDU AARE avant de générer la primitive .confirm, avec un résultat positif ou négatif.

¹² Pour prendre en charge des services de diffusion ou multidiffusion, une AA peut également être établie entre un AP de client et un groupe d'AP de serveur.

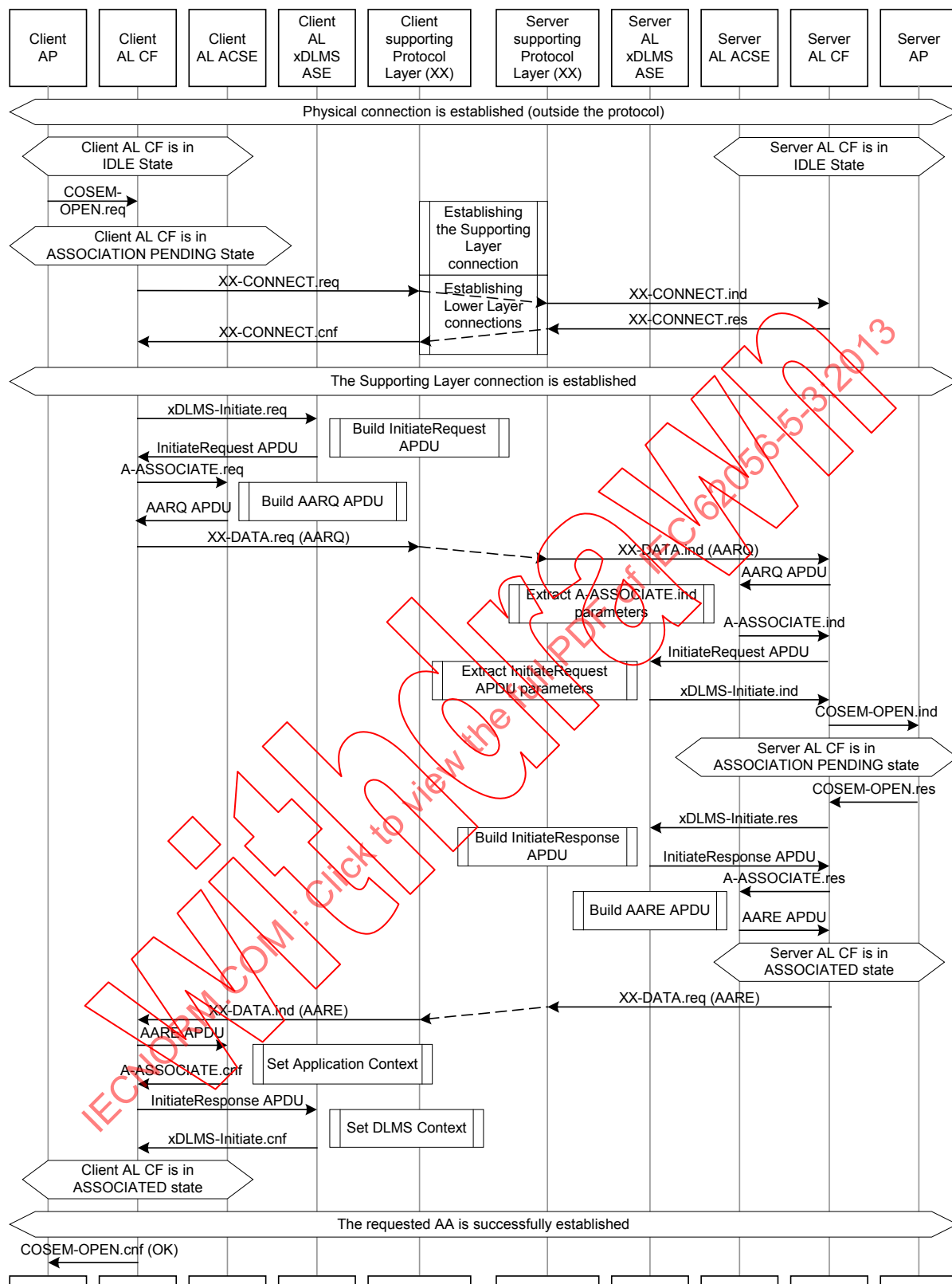
La CF du client passe à l'état ASSOCIATION PENDING. Elle examine ensuite le paramètre Protocol_Connection_Parameters. S'il indique que l'établissement de la connexion à la couche de support est requis, la CF établit la connexion¹³. La CF assemble alors –à l'aide de l'ACSE et de l'ASE xDLMS – l'APDU AARQ contenant les paramètres de la primitive COSEM-OPEN.request envoyés par l'AP et l'envoie ensuite au serveur.

La CF de l'AL de serveur indique l'APDU AARQ reçue à l'ACSE. Elle extrait les paramètres associés à l'ACSE, puis redonne le contrôle à la CF. La CF transfère ensuite le contenu du paramètre user-information de l'APDU AARQ - contenant une APDU xDLMS InitiateRequest - vers le xDLMS_ASE. Elle extrait les paramètres de cette APDU, puis redonne le contrôle à la CF. La CF génère la primitive COSEM-OPEN.indication sur l'AP du serveur avec les paramètres de l'APDU ¹⁴ reçue et passe à l'état ASSOCIATION PENDING.

NOTE Les ASE extraient uniquement les paramètres; leur interprétation et la décision d'accepter ou non l'AA proposée sont le rôle de l'AP du serveur.

¹³ La couche physique doit être connectée avant l'appel du service COSEM-OPEN.

¹⁴ Certains paramètres de service de la primitive COSEM-OPEN.indication (informations d'adresse, User_Information) ne proviennent pas de l'APDU AARQ mais de la trame de couche de support contenant l'APDU AARQ. Sous certains profils de communication, le paramètre Service_Class du service COSEM-OPEN est associé au type de trame de la couche de support. Dans d'autres profils de communication, il est associé au champ response-allowed de l'APDU xDLMS-Initiate.request. Voir également l'Annexe A.



Légende

IEC 1122/13

Anglais	Français
Client AP	AP du client
Client AL CF	CF d'AL du client
Client AL ACSE	ACSE d'AL du client
Client AL xDLMS ASE	ASE xDLMS d'AL du client

Anglais	Français
Client supporting Protocol Layer (XX)	Couche de protocole de support de client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support de serveur (XX)
Server AL xDLMS ASE	ASE xDLMS d'AL de serveur
Server AL ACSE	ACSE d'AL de serveur
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Physical connection is established (outside the protocole)	La connexion physique est établie (en dehors du protocole)
Client AL CF is in IDLE state	La CF d'AL du client est à l'état IDLE
Server AL CF is in IDLE state	La CF d'AL du serveur est à l'état IDLE
Client AL CF is in ASSOCIATION PENDING state	La CF d'AL du client est à l'état ASSOCIATION PENDING
Establishing the Supporting Layer connection	Établissement de la connexion à la couche de support
Establishing Lower Layer connections	Établissement des connexions à la couche inférieure
The Supporting Layer connection is established	La connexion à la couche de support est établie
InitiateRequest APDU	APDU InitiateRequest
Build InitiateRequest APDU	Générer l'APDU InitiateRequest
Build AARQ APDU	Générer l'APDU AARQ
AARQ APDU	APDU AARQ
Extract A-ASSOCIATE.ind parameters	Extraire les paramètres A-ASSOCIATE.ind
InitiateRequest APDU	APDU InitiateRequest
Extract InitiateRequest APDU parameters	Extraire les paramètres de l'APDU InitiateRequest
Server AL CF is ASSOCIATION PENDING state	La CF d'AL de serveur est à l'état ASSOCIATION PENDING
Build InitiateResponse APDU	Générer l'APDU InitiateResponse
InitiateReponse APDU	APDU InitiateResponse
Build AARE APDU	Générer l'APDU AARE
AARE APDU	APDU AARE
Server AL CF is in ASSOCIATED state	La CF d'AL de serveur est à l'état ASSOCIATED
AARE APDU	APDU AARE
Set Application Context	Définir le contexte d'application
InitiateResponse APDU	APDU InitiateResponse
Set DLMS Context	Définir le contexte DLMS
Client AL CF is in ASSOCIATED state	La CF d'AL de client est à l'état ASSOCIATED
The requested AA is successfully established	L'AA demandée est établie avec succès

Figure 11 – MSC pour l'établissement réussi d'une AA précédé de l'établissement réussi d'une connexion de couche inférieure de support

L'AP de serveur analyse les champs de l'APDU AARQ comme décrit ci-dessous.

Champs de l'unité fonctionnelle Kernel:

- application-context-name: contient le nom COSEM_Application_Context_Name proposé par le client pour l'association;

- **calling-AP-title:** lorsque le contexte d'application proposé utilise le chiffage, il doit contenir le titre de système client. Si un titre de système client a déjà été envoyé lors du processus d'enregistrement, par exemple dans le profil CPL S-FSK, le champ calling-AP-title doit contenir le même titre de système. Sinon, l'AA doit être rejetée et des informations de diagnostic appropriées doivent être envoyées;

Champs de l'unité fonctionnelle d'authentification (le cas échéant):

- **sender-acse-requirements:**
- si elle n'est pas présente ou si elle est présente mais que bit 0 = 0, l'unité fonctionnelle d'authentification n'est pas sélectionnée. Tous les champs suivants de l'unité fonctionnelle d'authentification sont susceptibles d'être ignorés;
- si elle est présente mais que bit 0 = 1, l'unité fonctionnelle d'authentification est sélectionnée.
- **mechanism-name:** contient le nom COSEM_Authentication_Mechanism_Name proposé par le client pour l'association;
- **calling-authentication-value:** contient la valeur d'authentification générée par le client.

Si la valeur des champs mechanism-name ou calling-authentication-value ne sont pas acceptables, l'AA proposée doit être refusée.

Lorsque l'analyse syntaxique des champs des unités fonctionnelles Kernel et d'authentification est terminée, le serveur passe à l'analyse des paramètres de l'APDU xDLMS InitiateRequest, contenus dans le champ user-information de l'AARQ:

- **dedicated-key:** contient la clé dédiée à utiliser dans l'AA en cours d'établissement;
- **response-allowed:** Si l'AA proposée est confirmée, la valeur de ce paramètre est TRUE (par défaut) et le serveur doit renvoyer une APDU AARE. Sinon, le serveur ne doit pas répondre. Voir également l'Annexe A.
 - a) **proposed-dlms-version-number.** Voir 4.2.3.1;
 - b) **proposed-conformance.** Voir 7.3.1;
 - c) **client-max-receive-pdu-size.** Voir 4.2.3.1.

Si tous les éléments de l'AA proposée sont acceptables, l'AP de serveur appelle la primitive de service COSEM-OPEN.response avec les paramètres suivants:

- **Application_Context_Name.** le même nom que celui proposé;
- **Responding-AP-title:** si le contexte d'application négocié utilise le chiffage, il doit contenir le titre du système serveur. Si un titre de système serveur a déjà été envoyé lors d'un processus d'enregistrement, comme dans le cas d'un profil CPL S-FSK, le paramètre Responding_AP_Title doit contenir le même titre de système. Sinon, le client doit annuler l'AA.
- **Result:** accepted;
- **Failure_Type:** Result-source: acse-service-user; Diagnostic: null;
- **User_Information:** une APDU xDLMS InitiateResponse contenant les éléments du contexte xDLMS négocié.

La CF assemble l'APDU AARE – à l'aide de l'ASE xDLMS et de l'ACSE – et l'envoie à l'AL du client via les protocoles de couche inférieure de support, puis passe à l'état ASSOCIATED. L'AA proposée est alors établie, le serveur peut recevoir des demandes de service de transfert de données xDLMS – confirmées et non-confirmées – et envoyer des réponses à ces demandes de service confirmées dans cette AA.

Côté client, les paramètres de l'APDU AARE reçus sont extraits à l'aide de l'ACSE et de l'ASE xDLMS puis sont transmis à l'AP du client via la primitive de service COSEM-OPEN.confirm.

Au même moment, l'AL du client passe à l'état ASSOCIATED. L'AA est alors établie avec le contexte d'application et le contexte xDLMS négociés.

Si le contexte d'application proposé par le client n'est pas acceptable ou si l'authentification du client échoue, la primitive COSEM-OPEN.response est appelée avec les paramètres suivants:

- Application_Context_Name: le même nom que celui proposé ou celui pris en charge par le serveur;
- Result: rejected-permanent ou rejected-transient;
- Failure_Type: Result-source: acse-service-user; Diagnostic: une valeur appropriée;
- User_Information: une APDU xDLMS InitiateResponse avec les paramètres du contexte xDLMS pris en charge par le serveur.

Si le contexte d'application proposé par le client est acceptable et si l'authentification du client réussit mais le contexte xDLMS ne peut pas être accepté, la primitive COSEM-OPEN.response doit être appelée avec les paramètres suivants:

- Application_Context_Name: le même nom que celui proposé;
- Result: rejected-permanent ou rejected-transient;
- Failure_Type: Result-source: acse-service-user; Diagnostic: no-reason-given;
- User_Information: une APDU xDLMS ConfirmedServiceError indiquant la raison du refus du contexte xDLMS proposé.

Dans ces deux cas, lors de l'appel d'une primitive .response, la CF assemble et envoie l'APDU AARE au client via les protocoles de couche inférieure de support. L'AA proposée n'est pas établie et la CF de serveur revient à l'état IDLE.

Côté client, les paramètres de l'APDU AARE reçus sont extraits à l'aide de l'ACSE et de l'ASE xDLMS puis sont transmis à l'AP du client via la primitive COSEM-OPEN.confirm. L'AA proposée n'est pas établie et la CF du client revient à l'état IDLE.

Il est possible que l'ACSE de serveur ne puisse pas prendre en charge l'association demandée, par exemple si la syntaxe AARQ ou la version de protocole ACSE ne sont pas acceptables. Dans cette situation, il renvoie une primitive COSEM-OPEN.response au client avec le paramètre Result approprié. La valeur symbolique « ACSE service-provider » est attribuée comme il convient au paramètre Result Source. La primitive COSEM-OPEN.indication n'est pas émise. L'association n'est pas établie.

7.2.4.2 Appels de service COSEM-OPEN répétés

Si une primitive COSEM-OPEN.request est appelée par l'AP de client référençant une AA déjà établie, l'AL confirme localement et négativement cette demande au motif que l'AA demandée existe déjà. À noter que c'est toujours le cas pour des AA pré-établies, voir 7.2.4.4.

Si néanmoins une AL de serveur reçoit une APDU AARQ référençant une AA déjà existante, elle supprime simplement cette AARQ, ou peut répondre avec l'APDU facultative ExceptionResponse, si cette dernière est utilisée.

7.2.4.3 Établissement d'une association d'applications non confirmée

Un AP de client souhaitant établir une AA non confirmée appelle la primitive COSEM-OPEN.request de l'ASO avec le paramètre Service_Class == Unconfirmed. Le paramètre response-allowed de l'APDU xDLMS InitiateRequest, contenu dans le paramètre user-information de l'AARQ est défini sur FALSE. L'AL du client n'attend aucune réponse du

serveur: la primitive .confirm est générée localement. Sinon, la procédure est identique à celle de l'établissement d'AA confirmées.

L'établissement d'AA non confirmées ne requiert pas la réponse de l'AP du serveur à la demande d'association venant du client; ainsi, dans certains cas, par exemple dans le cas de communications unidirectionnelles ou de diffusion - l'établissement d'AA non confirmées est la seule possibilité.

Lorsqu'une AA non confirmée est établie, les services de transfert de données xDLMS utilisant le référencement LN peuvent être appelés uniquement de façon non confirmée, jusqu'à la libération de l'association. Avec le référencement SN, seul le service UnconfirmedWrite peut être utilisé.

7.2.4.4 Associations d'applications pré-établies

Le but des AA pré-établies est de simplifier l'échange de données. Les phases d'établissement et de libération d'AA (phases 1 et 3 de la Figure 4) utilisant les services COSEM-OPEN et COSEM-RELEASE sont éliminées, et seuls les services de transfert de données sont utilisés¹⁵.

Cette norme n'indique pas comment établir ce type d'AA: on considère que cela a déjà été fait. Les AA pré-établies peuvent être considérées comme existantes dès que les couches inférieures peuvent transmettre des APDU entre le client et le serveur.

Les AA pré-établies peuvent être confirmées ou non-confirmées (en fonction de la façon dont elles ont été pré-établies).

Les AA pré-établies ne peuvent pas être libérées.

7.2.5 Protocole de libération d'association d'applications

7.2.5.1 Vue d'ensemble

Une AA existante peut être libérée avec ou sans perte de données. La libération sans perte de données est déclenchée par l'AP du client. La libération avec perte de données se produit lorsqu'un événement inattendu survient, par exemple lors de la détection d'une déconnexion physique.

7.2.5.2 Libération sans perte de données d'une association d'applications

DLMS/COSEM propose deux mécanismes de libération d'AA:

- déconnexion de la couche de support de l'AL;
- utilisation du service A-Release de l'ACSE.

Le premier mécanisme doit être pris en charge sous tous les profils où la couche de support de l'AL est orientée vers la connexion.

EXEMPLE Profil à 3 couches orienté sur la connexion basé sur HDLC.

Pour libérer une AA de cette façon, le service COSEM-RELEASE doit être appelé avec le paramètre Use_RLRQ_RLRE non présent ou défini sur FALSE. Lorsque la couche de support se déconnecte, toutes les AA générées sur cette connexion de couche de support doivent être libérées.

¹⁵ Comme pour toutes les AA, les dispositifs logiques doivent aussi contenir un objet d'interface Association LN / SN pour les associations pré-établies.

Le deuxième mécanisme peut être utilisé pour libérer une AA sans déconnecter la couche de support. Il doit être pris en charge sous tous les profils lorsque la couche de support n'est pas connectée. Il peut également être utilisé lorsque la couche de support est orientée vers la connexion mais la connexion n'est pas gérée par l'AL, ou lorsque la déconnexion de la couche de support n'est pas effective car d'autres applications l'utilisent, ou lorsque le service COSEM-RELEASE doit être sécurisé. C'est la seule façon de libérer des AA non confirmées.

Pour libérer une AA de cette façon, le service COSEM-RELEASE doit être appelé avec le paramètre `Use_RLRQ_RLRE` défini sur `TRUE`. Comme indiqué dans le paragraphe 6.3, le service COSEM-RELEASE peut être sécurisé en insérant une primitive `xDLMS InitiateRequest / InitiateResponse` chiffrée dans le champ `user-information` des APDU `RLRQ / RLRE` respectivement, empêchant ainsi un refus potentiel de service d'attaque.

Un exemple de libération d'une AA à l'aide du service A-RELEASE d'ACSE est fourni dans la Figure 12.

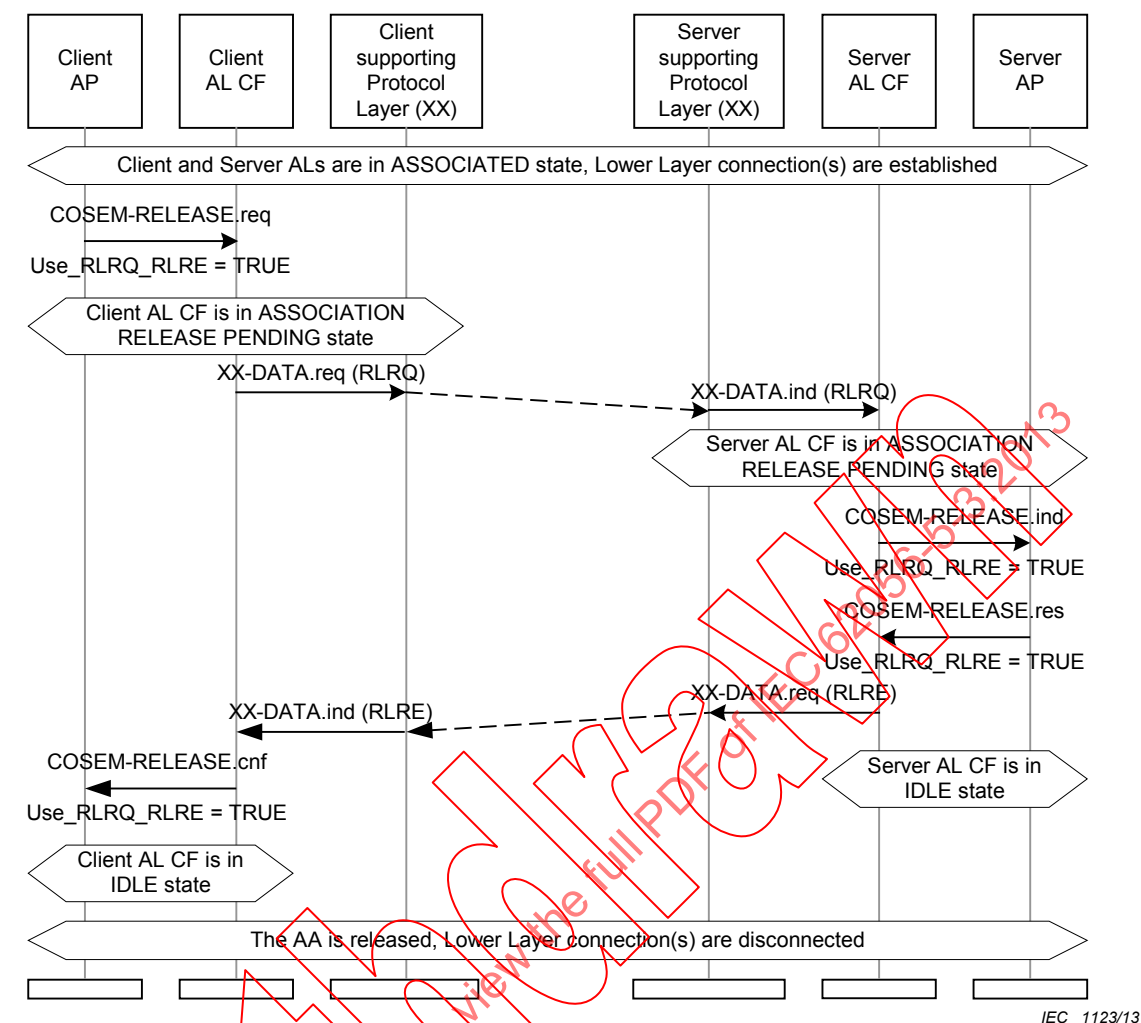
Un AP de client souhaitant libérer une AA à l'aide du service A-RELEASE doit appeler la primitive de service `COSEM-RELEASE.request` avec le paramètre `Use_RLRQ_RLRE == TRUE`. La CF du client passe à l'état `ASSOCIATION RELEASE PENDING`. Elle génère ensuite une APDU `RLRQ` et l'envoie au serveur. Si l'AA à libérer a été établie dans un contexte chiffré, le champ `user-information` de l'APDU `RLRQ` est susceptible de contenir une APDU `xDLMS InitiateRequest` chiffrée, voir 6.3.

Lorsque la CF d'AL de serveur reçoit une APDU `RLRQ`, elle contrôle d'abord si le champ `user-information` contient une APDU `xDLMS InitiateRequest` chiffrée. Si oui, elle tente de la déchiffrer. Si elle réussit, elle passe à l'état `ASSOCIATION RELEASE PENDING` et génère une primitive `COSEM-RELEASE.indication` avec le paramètre `Use_RLRQ_RLRE == TRUE`. Sinon, elle supprime en mode silencieux l'APDU `RLRQ` et reste à l'état `ASSOCIATED`.

La primitive `.response` est appelée par l'AP de serveur pour indiquer si la libération de l'AA est acceptée ou non, mais uniquement si l'AA à libérer est confirmée. À noter que l'AP de serveur ne peut pas refuser une demande de libération. À réception de la primitive `.response`, la CF de l'AL de serveur génère une APDU `RLRE` et l'envoie au client. Si l'APDU `RLRQ` contient une APDU `xDLMS InitiateRequest` chiffrée, l'APDU `RLRE` doit contenir une APDU `xDLMS InitiateResponse` chiffrée. La CF de l'AL de serveur revient à l'état `IDLE`.

La primitive `.confirm` est générée par la CF de l'AL du client à la réception de l'APDU `RLRE`. La couche de support n'est pas déconnectée. La CF de l'AL du client revient à l'état `IDLE`.

Si l'APDU `RLRE` reçue contient une APDU `xDLMS InitiateResponse` chiffrée mais ne peut pas être déchiffrée, l'APDU `RLRE` doit être supprimée. Le client doit gérer la situation.



Légende

Anglais	Français
Client AP	AP du client
Client AL CF	CF d'AL de client
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support du serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs are in ASSOCIATED state, Lower Layer connection(s) are established	Les AL de serveur et de client sont à l'état ASSOCIATED, la ou les connexions de la couche inférieure sont établies
Client AL CF is in ASSOCIATION RELEASE PENDING state	La CF d'AL du client est à l'état ASSOCIATION RELEASE PENDING
Server AL CF is in ASSOCIATION RELEASE PENDING state	La CF d'AL de serveur est à l'état ASSOCIATION RELEASE PENDING
Server AL CF is in IDLE state	La CF d'AL du serveur est à l'état IDLE
Client AL CF is in IDLE state	La CF d'AL du client est à l'état IDLE
The AA is released, Lower Layer connection(s) are disconnected	L'AA est libérée, la ou les connexions de la couche inférieure sont désactivées

Figure 12 – Libération d'AA sans perte de données à l'aide du service A-RELEASE

La Figure 13 fournit un exemple de libération sans perte de données d'une AA confirmée, via la déconnexion de la couche inférieure correspondante.

Un AP de client souhaitant libérer une AA sans utiliser le service A-RELEASE appelle la primitive COSEM-RELEASE.request avec le paramètre Use_RLRQ_RLRE == FALSE ou non présent. La CF d'AL de client passe à l'état ASSOCIATION RELEASE PENDING.

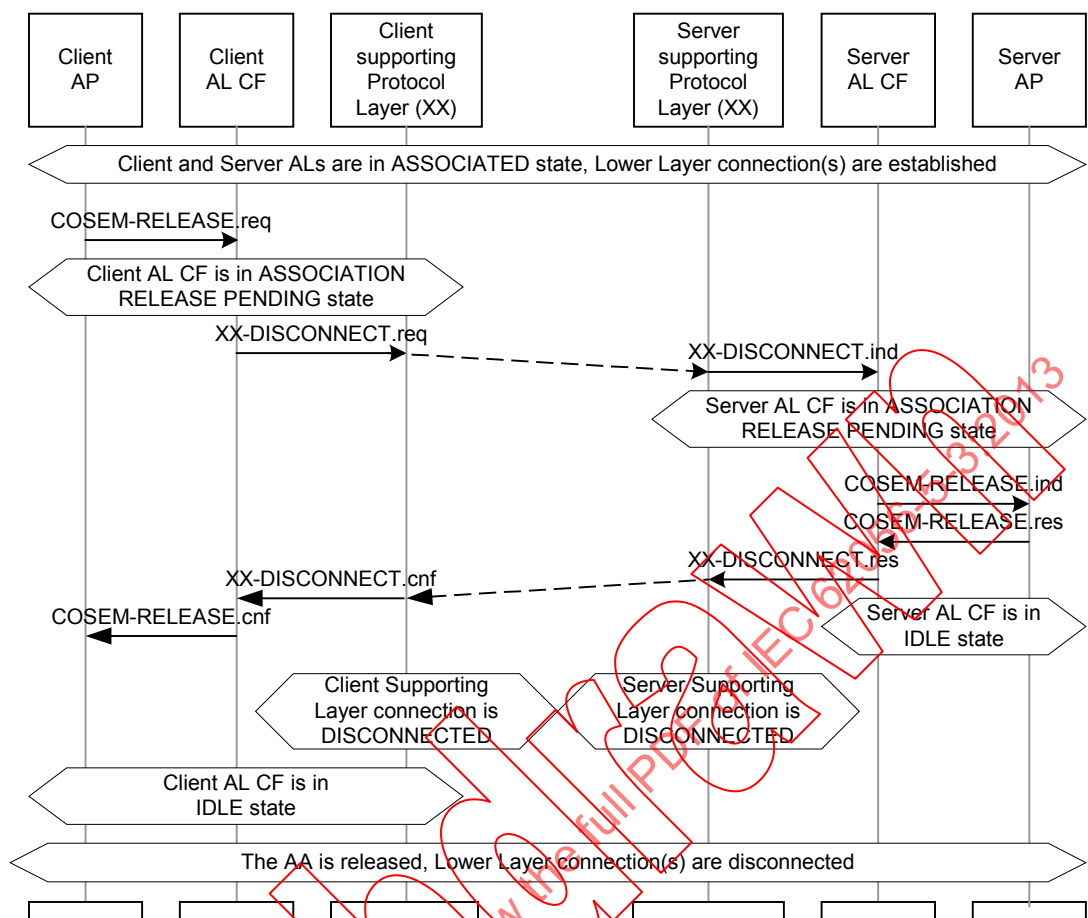
Sous les profils de communication où le service RLRQ est obligatoire, le fait d'appeler la primitive .request avec le paramètre Use_RLRQ_RLRE absent ou avec le paramètre Use_RLRQ_RLRE == FALSE peut générer une erreur: la primitive .request doit être refusée en local. La CF de l'AL du client revient à l'état IDLE.

Lorsque la CF de l'AL de client reçoit la primitive .request, elle envoie une primitive XX-DISCONNECT.request au serveur.

Lorsque la CF d'AL de serveur reçoit la primitive XX-DISCONNECT.request, la CF passe à l'état ASSOCIATION RELEASE PENDING. La primitive COSEM-RELEASE.indication est générée par la CF d'AL de serveur avec le paramètre Use_RLRQ_RLRE non présent ou défini sur FALSE.

La primitive COSEM-RELEASE.response est appelée par l'AP de serveur pour indiquer si la libération de l'AA est acceptée ou non. À noter que l'AP de serveur ne peut pas refuser une demande de libération. À réception de cette primitive, la CF d'AL du serveur envoie la primitive XX-DISCONNECT.response au client et revient à l'état IDLE.

La primitive COSEM-RELEASE.confirm est générée par l'AL du client à réception de la primitive XX-DISCONNECT.confirm. La couche de support est déconnectée. La CF de l'AL du client revient à l'état IDLE.



IEC 1124/13

Légende

Anglais	Français
Client AP	AP du client
Client AL CF	CF d'AL du client
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support du serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs are in ASSOCIATED state, Lower Layer connection(s) are established	Les AL de serveur et de client sont à l'état ASSOCIATED, la ou les connexions de la couche inférieure sont établies
Client AL CF is in ASSOCIATION RELEASE PENDING state	La CF d'AL du client est à l'état ASSOCIATION RELEASE PENDING
Server AL CF is in ASSOCIATION RELEASE PENDING state	La CF d'AL de serveur est à l'état ASSOCIATION RELEASE PENDING
Server AL CF is in IDLE state	La CF d'AL du serveur est à l'état IDLE
Client Supporting Layer connection is DISCONNECTED	La connexion de la couche de support du client est DISCONNECTED
Server Supporting Layer connection is DISCONNECTED	La connexion de la couche de support du serveur est DISCONNECTED
Client AL CF is in IDLE state	La CF d'AL du client est à l'état IDLE
The AA is released, Lower Layer connection(s) are disconnected	L'AA est libérée, la ou les connexions à la couche inférieure sont désactivées.

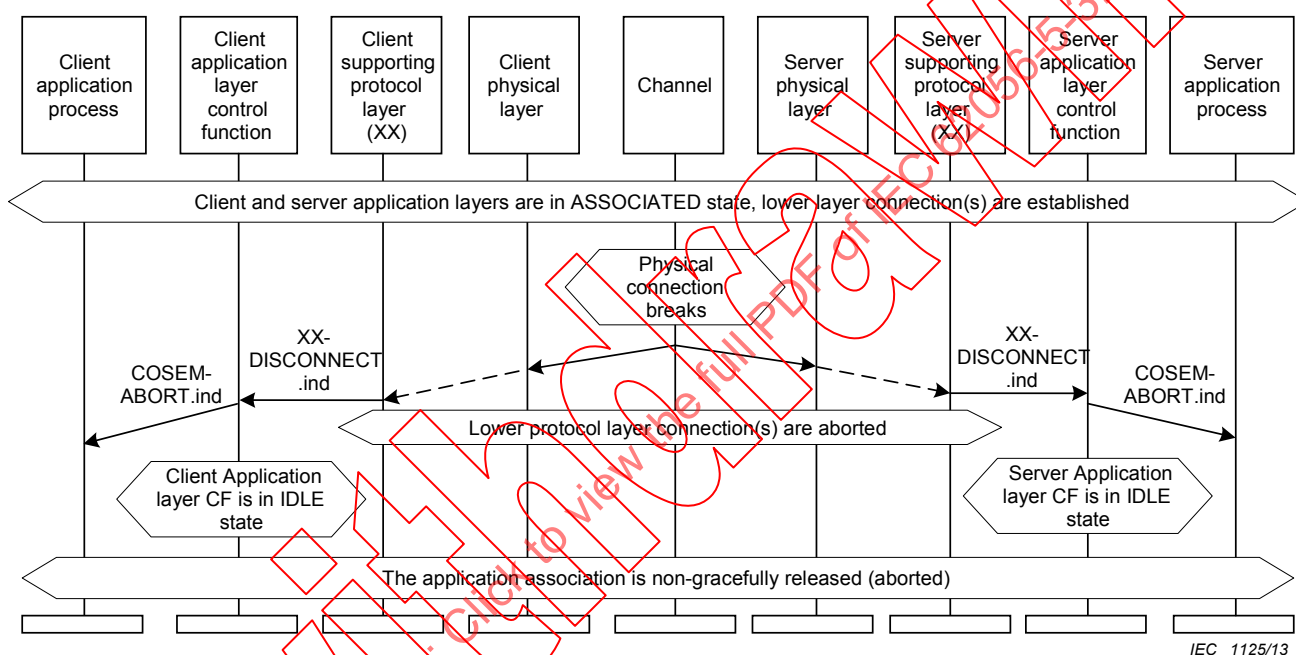
Figure 13 – Libération d'AA sans perte de données par déconnexion de la couche de support

7.2.5.3 Libération avec perte de données d'une association d'applications

Plusieurs événements peuvent entraîner une libération d'AA avec perte de données: la détection d'une déconnexion d'une couche inférieure (notamment la connexion physique), la détection d'une erreur locale, etc.

La libération avec perte de données – l'abandon – d'une AA est signalée à l'AP COSEM via le service COSEM-ABORT. Le paramètre Diagnostics de ce service indique la raison de la libération d'AA avec perte de données. La libération avec perte de données d'une AA n'est pas sélective: si elle se produit, toutes les associations existantes (à l'exception de celles pré-établies) doivent être abandonnées.

La Figure 14 inclut le tableau de séquences de message générées par l'abandon d'une AA en raison de la détection d'une déconnexion physique.



Légende

Anglais	Français
Client application process	Processus d'application du client
Client application layer control function	Fonction de contrôle de la couche d'application du client
Client supporting protocol layer (XX)	Couche de protocole de support du client (XX)
Client physical layer	Couche physique du client
Channel	Canal
Server physical layer	Couche physique du serveur
Server supporting protocol layer (XX)	Couche de protocole de support du serveur (XX)
Server application layer control function	Fonction de contrôle de couche d'application de serveur
Server application process	Processus d'application du serveur
Client and server application layers are in ASSOCIATED state, lower layer connection(s) are established	Les couches d'application du client et du serveur sont à l'état ASSOCIATED, la ou les connexions de la couche inférieure sont établies
Physical connection breaks	Interruptions de la connexion physique
Lower protocol layer connection(s) are aborted	La ou les connexions de la couche de protocole inférieure sont abandonnées

Anglais	Français
Client layer CF is in IDLE state	La CF de la couche d'application du client est à l'état IDLE
Server layer CF is in IDLE state	La CF de la couche d'application du serveur est à l'état IDLE
The application association is non-gracefully released	L'association d'applications est libérée avec perte de données

Figure 14 – Abandon d'une AA suite à la primitive PH-ABORT.indication ¹⁶

7.3 Protocole des services de transfert de données

7.3.1 Négociation de services et d'options - Bloc de conformité

Le bloc de conformité permet aux clients et aux serveurs utilisant le même protocole DLMS/COSEM, mais prenant en charge des options différentes, de négocier un ensemble d'options de communication compatible. Il est contenu dans le paramètre DLMS_Conformance du service COSEM-OPEN.

Dans DLMS/COSEM, aucun service, ni aucune option n'est obligatoire: ceux utilisés sont négociés via le service COSEM-OPEN (le paramètre proposed-conformance de l'APDU xDLMS InitiateRequest et le paramètre negotiated-conformance de l'APDU xDLMS InitiateResponse). Un service implémenté doit être totalement conforme à ces spécifications. Si un service ou une option n'est pas présent(e) dans le bloc de conformité négocié, il/elle peut ne pas être demandé(e) par le client.

Le bloc de conformité xDLMS peut être différencié du bloc de conformité DLMS indiqué dans la CEI 61334-4-41 grâce à cette étiquette: APPLICATION 31. Voir le Tableau 35.

Tableau 35 – Bloc de conformité xDLMS

Bit de bloc de conformité	Réservé	Référencement LN	Référencement SN
0	x		
1	x		
2	x		
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 La libération d'une AA peut nécessiter une communication interne entre les ASE de la CF. Cela n'est pas indiqué dans les figures.

Bit de bloc de conformité	Réservé	Référencement LN	Référencement SN
16	x		
17	x		
18			parameterized-access
19		get	
20		Set	
21		selective-access	
22		event-notification	
23		action	

7.3.2 Appels de service confirmés et non confirmés

Généralement, les services de transfert de données peuvent être appelés de façon confirmée ou non confirmée. La séquence chronologique des primitives de service est la suivante:

- Figure 8 point a) pour les appels de service confirmés; et
- Figure 8 point d) pour les appels de service non confirmés.

Une AP du client souhaitant accéder à un attribut ou à une méthode d'interface d'objet COSEM appelle la primitive de service .request appropriée. L'AL du client génère l'APDU correspondant à la primitive .request et l'envoie au serveur.

À la réception de la primitive .indication, l'AP du serveur vérifie si le service peut être fourni ou non (validité, droits d'accès du client, disponibilité, etc.). Si tout est correct, elle applique le service requis sur l'objet « réel » correspondant. Dans le cas de services confirmés, l'AP de serveur appelle la primitive .response appropriée. L'AL du serveur génère l'APDU correspondant à la primitive .response et l'envoie au serveur. L'AL du client génère la primitive .confirm.

Si une demande de service confirmée ne peut pas être traitée par l'AL du serveur (par exemple si la demande a été reçue sans qu'une AA ait été préalablement établie ou si la demande présente une erreur) - soit elle est ignorée, soit l'AL du serveur y répond, si possible, avec une APDU ConfirmedServiceError ou une APDU ExceptionResponse, le cas échéant. Ces APDU peuvent contenir des informations de diagnostic sur la raison de l'échec du traitement de la demande. Elles sont définies à l'Article 8.

Dans les AA confirmées, les services de transfert de données de type client/serveur peuvent être appelés de manière confirmée ou non confirmée.

Dans les AA non confirmées, les services de transfert de données de type client/serveur peuvent être appelés de manière non confirmée uniquement. Les collisions dues à des réponses multiples possibles dans le cas d'une multidiffusion et/ou d'une diffusion peuvent ainsi être évitées.

Dans le cas de services non confirmés, trois types différents d'adresses de destination sont possibles: individuel, groupe ou diffusion. En fonction du type de l'adresse de destination, le poste destinataire doit gérer les APDU entrantes différemment, à savoir:

- XX-APDU avec l'adresse individuelle d'une unité logique COSEM. Si elles sont reçues dans une AA établie, elles doivent être envoyées au dispositif logique COSEM destinataire, sinon, elles doivent être ignorées;
- XX-APDU avec une adresse de groupe de dispositifs logiques COSEM. Elles doivent être envoyées au groupe de dispositifs logiques COSEM destinataire. Cependant, le message

reçu doit être ignoré s'il n'existe aucune AA établie entre un client et le groupe de dispositifs logiques COSEM destinataire;

- Les XX- APDU avec adresse de diffusion doivent être envoyées vers tous les dispositifs logiques COSEM destinataires. Cependant, le message reçu doit être ignoré si aucune AA n'est établie entre le client et l'adresse de l'ensemble des postes.

NOTE Des AA non confirmées établies entre un client et un groupe d'unités logiques sont établies avec un service COSEM-OPEN avec Service_Class == Unconfirmed et un groupe d'adresses de dispositifs logiques (par exemple, adresse de diffusion).

7.3.3 Protocole du service GET

Lorsque l'AP du client souhaite lire la valeur d'un ou plusieurs attributs d'objet d'interface COSEM, il utilise le service GET.

Comme expliqué en 6.6, la forme codée de la demande doit toujours être contenue dans une seule APDU.

Cependant, il est possible que le résultat soit trop long pour figurer dans une seule APDU. Dans ce cas, le transfert de bloc peut être utilisé. Il est négocié via le bit 11 du bloc de conformité; voir 7.3.1.

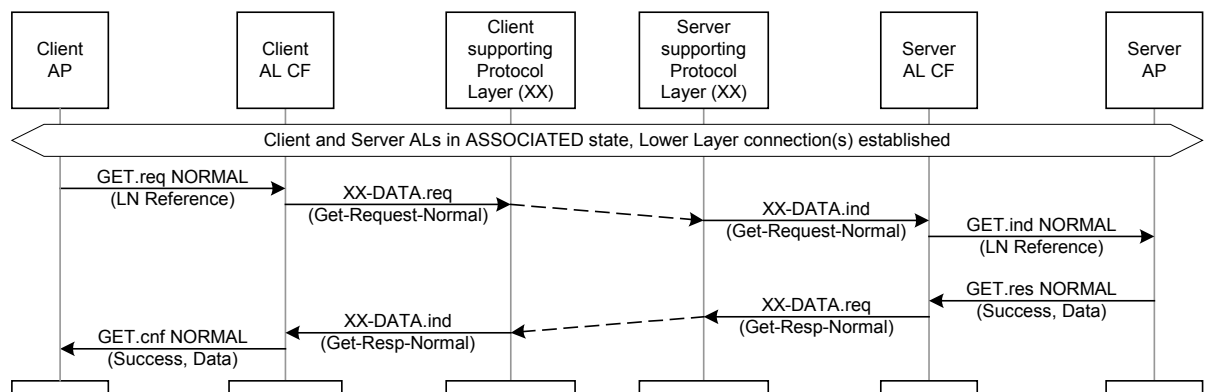
NOTE 1 La couche de support peut également fournir un mécanisme de transfert de messages longs.

Les types de primitive de service GET et les APDU correspondantes figurent dans le Tableau 36.

Tableau 36 – Types et APDU de service GET

GET .req / .ind	APDU Request	APDU Response	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

La Figure 15 indique le MSC pour un service GET confirmé en cas de succès, sans transfert de bloc.



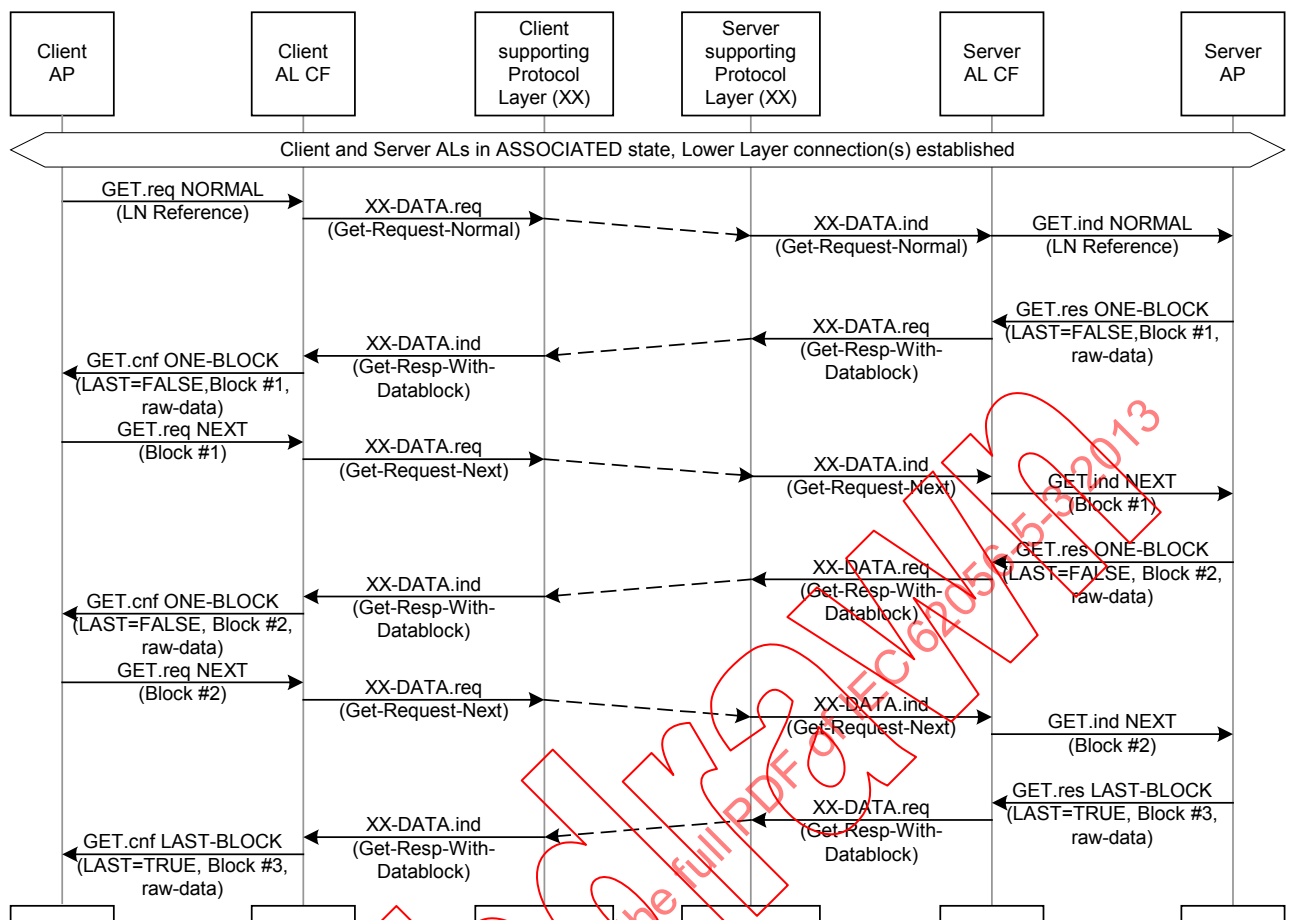
IEC 1126/13

Légende

Anglais	Français
Client AP	AP de client
Client AL CF	CF d'AL de client
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support de serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs in ASSOCIATED state, Lower Layer connection(s) established	Les AL de client et de serveur sont à l'état ASSOCIATED, la ou les connexions de la couche inférieure sont établies
LN Reference	Référence LN

Figure 15 – MSC du service GET

La Figure 16 indique le MSC d'un service GET confirmé en cas de succès; le résultat est retourné en trois blocs.



IEC 1127/13

Légende

Anglais	Français
Client AP	AP de client
Client AL CF	CF d'AL de client
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support de serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs in ASSOCIATED state, Lower Layer connection(s) established	Les AL de client et de serveur sont à l'état ASSOCIATED, la ou les connexions de la couche inférieure sont établies
LN Reference	Référence LN

Figure 16 – MSC du service GET avec transfert de bloc

La primitive de service GET.request est appelée avec le type Request_Type == NORMAL ou WITH-LIST, comme il convient. Dans ce cas, les données devant être retournées sont trop longues pour être contenues dans une seule APDU. L'AP de serveur les envoie donc en plusieurs blocs. Les données sont d'abord codées, comme si elles pouvaient être contenues dans une seule APDU:

- si la valeur d'un attribut unique a été demandée, seuls le type et la valeur doivent être codés (Data). Si l'attribut Data ne peut pas être fourni, la réponse doit être de type GET-NORMAL avec l'attribut Data_Access_Result;
- si la valeur d'une liste d'attributs a été demandée, la liste de résultats doit être codée pour chaque attribut (Data ou Data_Access_Result).

Le résultat est une série d'octets, $B_1, B_2, B_3, \dots, B_N$.

NOTE 2 Le serveur peut générer la réponse complète à réception de la première primitive GET.indication ou de façon dynamique (à la volée).

L'AP de serveur assemble ensuite la structure DataBlock_G:

- Last_Block == FALSE;
- Block_Number == 1;
- Result (Raw_Data) == les premiers octets K des données codées: $B_1, B_2, B_3, \dots, B_K$.

Il est recommandé de commencer la numérotation des blocs à partir de 1.

L'AP de serveur appelle alors la primitive de service GET-RESPONSE-ONE-BLOCK avec le type Response_Type == ONE-BLOCK contenant cette structure DataBlock-G.

À réception de la primitive .response, l'AL de serveur génère une APDU Get-Response-With-Datablock contenant les paramètres de la primitive .response et l'envoi au client.

À réception de cette APDU, l'AL de client génère une primitive confirm avec le type Response_Type == ONE-BLOCK. L'AP du client est alors informée que la réponse sera fournie en plusieurs blocs. Elle stocke le bloc de données reçu ($B_1, B_2, B_3, \dots, B_K$), puis accuse réception des données et demande le bloc suivant en appelant une primitive de service GET-REQUEST-NEXT. Le numéro de bloc doit être identique au numéro du bloc de données reçu. L'AL du client génère ensuite une APDU Get-Request-Next et l'envoi au serveur.

Lorsque l'AL de serveur appelle la primitive GET.indication avec le type Request_Type == NEXT, l'AP du serveur prépare et envoie le bloc de données suivant, notamment $B_{K+1}, B_{K+2}, B_{K+3}, \dots, B_L$, avec le paramètre block-number == 2. Cet échange bloc de données-accusés de réception se poursuit jusqu'à l'envoi du dernier bloc, contenant $B_M, B_{M+1}, B_{M+2}, \dots, B_N$. La dernière primitive GET response est appelée avec le paramètre Response_Type == LAST-BLOCK: dans la structure DataBlock-G, Last_Block == TRUE. Le client n'accuse pas réception de ce dernier bloc de données.

Tout au long de la procédure, les paramètres Invoke_Id et Priority doivent être identiques dans chaque primitive.

Si lors d'un transfert de données long, le serveur reçoit une autre demande de service, celle-ci est traitée en fonction des règles de priorité et des paramètres de gestion des priorités (Bit 9 du bloc de conformité).

Si une erreur se produit lors d'un transfert de données long, le transfert est annulé. Les situations d'erreur sont les suivantes:

- a) Le serveur ne peut pas fournir le bloc de données suivant, pour une raison quelconque. Dans ce cas, l'AP du serveur doit appeler une primitive de service GET-RESPONSE-LAST-BLOCK. Le paramètre Result doit contenir une structure DataBlock_G avec:
 - Last_Block == TRUE;
 - Block_Number == numéro de bloc confirmé par le client +1;
 - Result == Data_Access_Result, indiquant la raison de l'échec.
- b) Le paramètre Block_Number d'une primitive de service GET-REQUEST-NEXT n'est pas égal au numéro du bloc précédent envoyé par le serveur. Le serveur interprète cette erreur comme si le client souhaitait annuler le transfert en cours. Dans ce cas, l'AP de serveur doit appeler une primitive de service GET-RESPONSE-LAST-BLOCK. Le paramètre Result doit contenir une structure DataBlock_G avec:

- Last_Block == TRUE;
- Block_Number == égal au numéro du bloc reçu par le client;
- Result == Data-Access-Result, long-get-aborted.

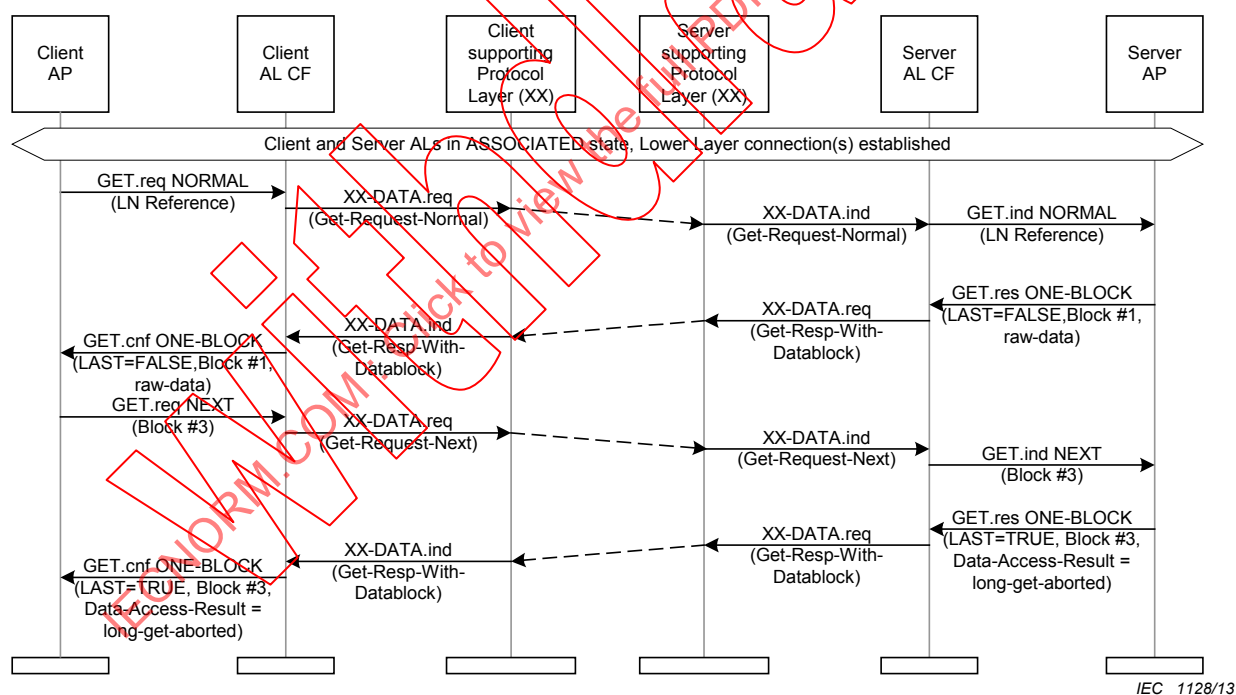
c) Il est possible que le serveur reçoive une APDU Get-Request-Next lorsqu'aucun transfert de données long n'est en cours. Dans ce cas, l'AP du serveur doit appeler une primitive de service GET-RESPONSE-LAST-BLOCK. Le paramètre Result doit contenir une structure DataBlock_G avec:

- Last-block == TRUE;
- Block-Number == égal au numéro de bloc envoyé par le client;
- Result == Data-Access-Result, no-long-get-in-progress.

d) Le numéro de bloc envoyé par le serveur n'est pas égal au numéro suivant de la séquence. Dans ce cas, le client doit abandonner le transfert de bloc (voir cas b).

Si, lors des situations d'erreur décrites ci-dessus, le serveur ne peut pas appeler une primitive de service GET-RESPONSE-LAST-BLOCK pour quelque raison que ce soit, il doit appeler une primitive de service GET-RESPONSE-NORMAL, avec le paramètre Data-Access-Result indiquant la raison de l'échec. Le serveur doit envoyer une APDU Get-Response-Normal.

Le MSC de la situation d'erreur b), transfert long abandonné, apparaît dans la Figure 17:



Légende

Anglais	Français
Client AP	AP de client
Client AL CF	CF d'AL de client
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support de serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs in ASSOCIATED state, Lower Layer connection(s) established	Les AL de client et de serveur sont à l'état ASSOCIATED, la ou les connexions de la couche

Anglais	Français
	inférieure sont établies
LN Reference	Référence LN

Figure 17 – MSC du service GET avec transfert de bloc, GET long abandonné

7.3.4 Protocole du service SET

Lorsque l'AP du client souhaite écrire la valeur d'un ou plusieurs attributs d'objet d'interface COSEM, il utilise le service SET.

Comme expliqué en 6.7, la forme codée de la demande peut être contenue dans une seule demande, ou non. Dans ce cas, le transfert par bloc peut être utilisé. Il est négocié via le bit 12 du bloc de conformité; voir 7.3.1.

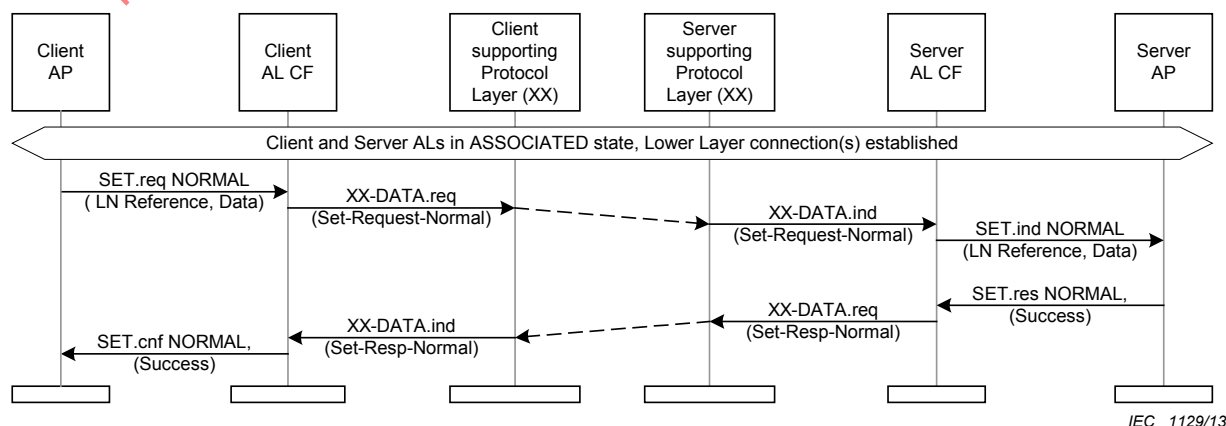
NOTE 1 La couche de support peut également fournir un mécanisme de transfert de messages longs.

Les types de primitive de service SET et les APDU correspondantes figurent dans le Tableau 37.

Tableau 37 – Types et APDU de service SET

SET .req / .ind	APDU Request	APDU Response	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
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

La Figure 18 inclut le MSC du service SET confirmé, en cas de succès et sans transfert de bloc.

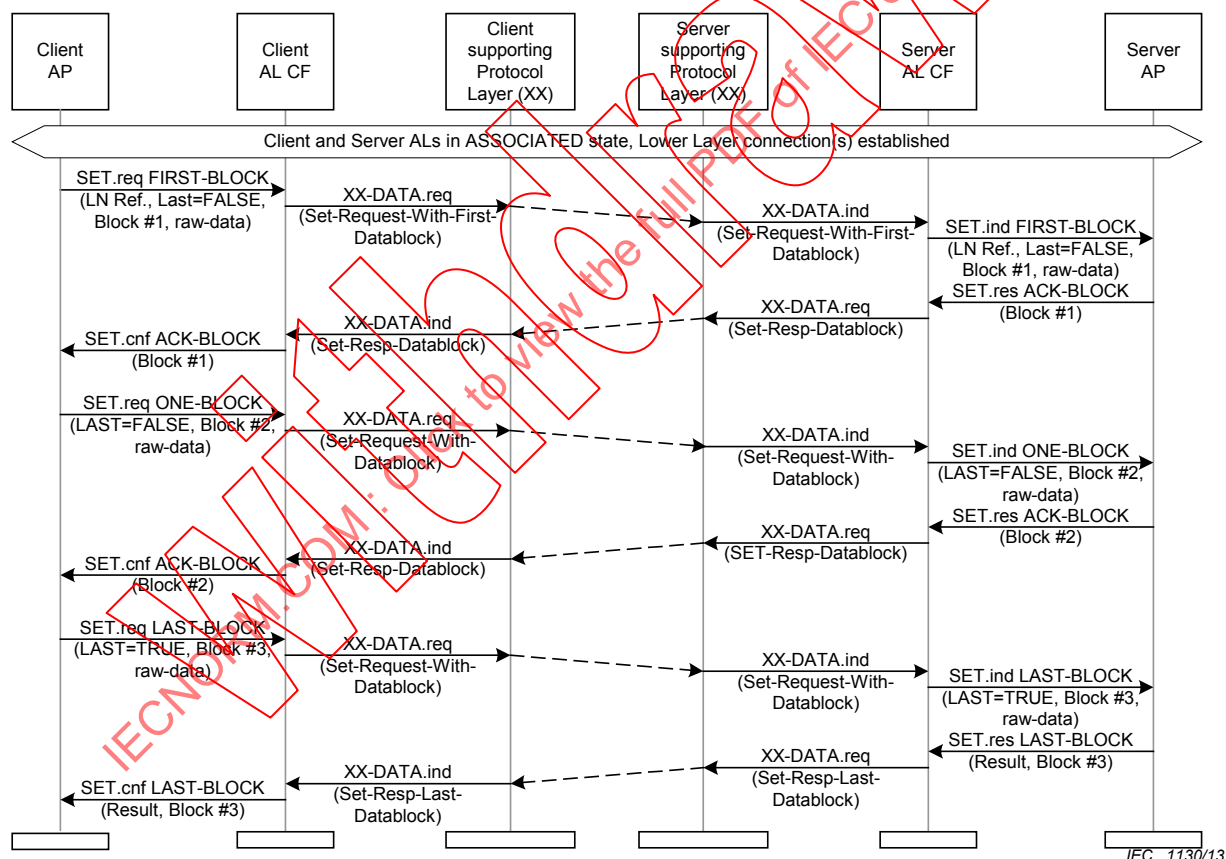


Légende

Anglais	Français
Client AP	AP de client
Client AL CF	CF d'AL de client
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support de serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs in ASSOCIATED state, Lower Layer connection(s) established	Les AL de client et de serveur sont à l'état ASSOCIATED, la ou les connexions de la couche inférieure sont établies
LN Reference	Référence LN

Figure 18 – MSC du service SET

La Figure 19 montre le MSC d'un service SET confirmé en cas de succès, avec envoi de la demande en trois blocs.



IEC 1130/13

Légende

Anglais	Français
Client AP	AP de client
Client AL CF	CF d'AL de client
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support de serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs in ASSOCIATED state,	Les AL de client et de serveur sont à l'état

Anglais	Français
Lower Layer connection(s) established	ASSOCIATED, la ou les connexions de la couche inférieure sont établies
LN Ref	Réf. LN

Figure 19 – MSC du service SET avec transfert de bloc

Dans ce cas, les données à envoyer sont trop longues pour rentrer dans une seule APDU. L'AP du client les envoie donc en plusieurs blocs. Les données sont d'abord codées, comme si elles pouvaient être contenues dans une seule APDU. Le résultat est une série d'octets, $B_1, B_2, B_3, \dots, B_N$.

NOTE 2 Le client peut générer la demande complète ($B_1, B_2, B_3, \dots, B_N$) en une étape, ou de façon dynamique (à la volée).

L'AP du client assemble ensuite la structure DataBlock_SA:

- Last_Block == FALSE;
- Block_Number == 1;
- Raw_Data == les premiers octets K de données codées: $B_1, B_2, B_3, \dots, B_K$.

L'AP de client appelle alors, selon qu'il convient, la primitive de service SET-REQUEST-FIRST-BLOCK ou SET-REQUEST-FIRST-BLOCK-WITH-LIST, qui contient la ou les références d'attribut et la structure DataBlock-SA.

À réception de la primitive .request, l'AL de client génère l'APDU Set-Request appropriée contenant les paramètres de la primitive .request et l'envoie au serveur.

Le serveur stocke le bloc de données reçu, puis en accuse réception et demande le bloc suivant en appelant une primitive SET.response avec le paramètre Response_Type == ACK-BLOCK et avec le même numéro de bloc que celui du bloc reçu.

Pour envoyer le bloc de données suivant contenant $B_{K+1}, B_{K+2}, B_{K+3}, \dots, B_L$, l'AP de client appelle une primitive de service SET-REQUEST-ONE-BLOCK. Cet échange blocs de données-accusés de réception se poursuit jusqu'à l'envoi du dernier bloc de données contenant $B_M, B_{M+1}, B_{M+2}, \dots, B_N$, par l'appel de la primitive de service SET-REQUEST-LAST-BLOCK avec le paramètre Last_Block == TRUE.

Lorsque ces primitives sont appelées, l'AL du client génère une APDU Set-Request-With-Datablock contenant une structure DataBlock_SA et envoie ces APDU au serveur.

Lorsque l'AP du serveur reçoit le dernier bloc de données, elle appelle une primitive de service SET-RESPONSE-LAST-BLOCK ou SET-RESPONSE-LAST-BLOCK-WITH-LIST, comme il convient. Le paramètre Result contient le résultat de l'appel de service SET complet. Le paramètre Block_Number confirme la réception du dernier bloc.

Tout au long de la procédure, les paramètres Invoke_Id et Priority doivent être identiques dans chaque primitive.

Si lors d'un transfert de données long, le serveur reçoit une autre demande de service, celle-ci est traitée en fonction des règles de priorité et des paramètres de gestion des priorités (Bit 9 du bloc de conformité).

Si une erreur se produit lors d'un transfert de données long, le transfert est annulé. Les situations d'erreur sont les suivantes:

- a) Le serveur ne peut pas gérer le bloc de données reçu, pour une raison quelconque. Dans ce cas, l'AP de serveur doit appeler une primitive de service SET-RESPONSE-LAST-

BLOCK ou SET-RESPONSE-LAST-BLOCK-WITH-LIST, comme il convient. Le paramètre Result indique la raison de l'abandon du transfert;

- b) Le paramètre Block_Number d'une primitive de service SET-REQUEST-ONE-BLOCK n'est pas égal au numéro de bloc attendu par le serveur (dernier reçu + 1). Le serveur interprète cette erreur comme si le client souhaitait annuler le transfert en cours. Dans ce cas, l'AP de serveur doit appeler une primitive de service SET-RESPONSE-LAST-BLOCK ou SET-RESPONSE-LAST-BLOCK-WITH-LIST comme il convient, avec le paramètre Result défini comme suit: Data_Access_Result == long-set-aborted;
- c) Il est possible que le serveur reçoive une APDU Set-Request-With-Datablock lorsqu'aucun transfert de données long n'est en cours. Dans ce cas, l'AP de serveur doit appeler une primitive de service SET-RESPONSE-LAST-BLOCK avec le paramètre Result défini comme suit: Data_Access_Result == no-long-set-in-progress.

Si, dans les situations d'erreur décrites ci-dessus, le serveur ne peut pas appeler de primitive de service SET-RESPONSE-LAST-BLOCK pour une raison quelconque, il appelle une primitive de service SET-RESPONSE-NORMAL avec le paramètre Data_Access_Result indiquant la raison de l'échec.

7.3.5 Protocole du service ACTION

Lorsque l'AP du client souhaite appeler une ou plusieurs méthodes d'objet d'interface COSEM, elle utilise le service ACTION. Comme expliqué en 6.8, le service ACTION se divise en deux phases.

Si les références de méthode et les paramètres d'appel de méthode ou paramètres de retour ne peuvent être contenus dans une seule APDU, le transfert par bloc peut être utilisé, du client vers le serveur et/ou du serveur vers le client. Il est négocié via le bit 13 du bloc de conformité; voir 7.3.1.

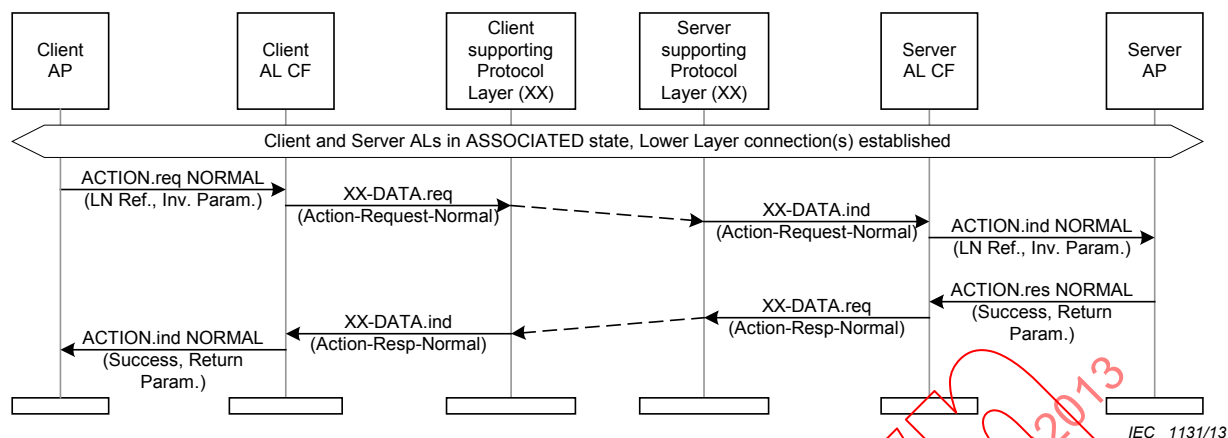
NOTE La couche de support peut également fournir un mécanisme de transfert de messages longs.

Les types de primitive de service ACTION et les APDU correspondantes figurent dans le Tableau 38.

Tableau 38 – Types et APDU de service ACTION

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

La Figure 20 montre le MSC du service ACTION confirmé, en cas de succès et sans transfert de bloc.



Légende

Anglais	Français
Client AP	AP de client
Client AL CF	CF d'AL de client
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support de serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs in ASSOCIATED state, Lower Layer connection(s) established	Les AL de client et de serveur sont à l'état ASSOCIATED, la ou les connexions de la couche inférieure sont établies
LN Ref	Réf. LN

Figure 20 – MSC du service ACTION

Le service ACTION peut transporter des données dans les deux directions:

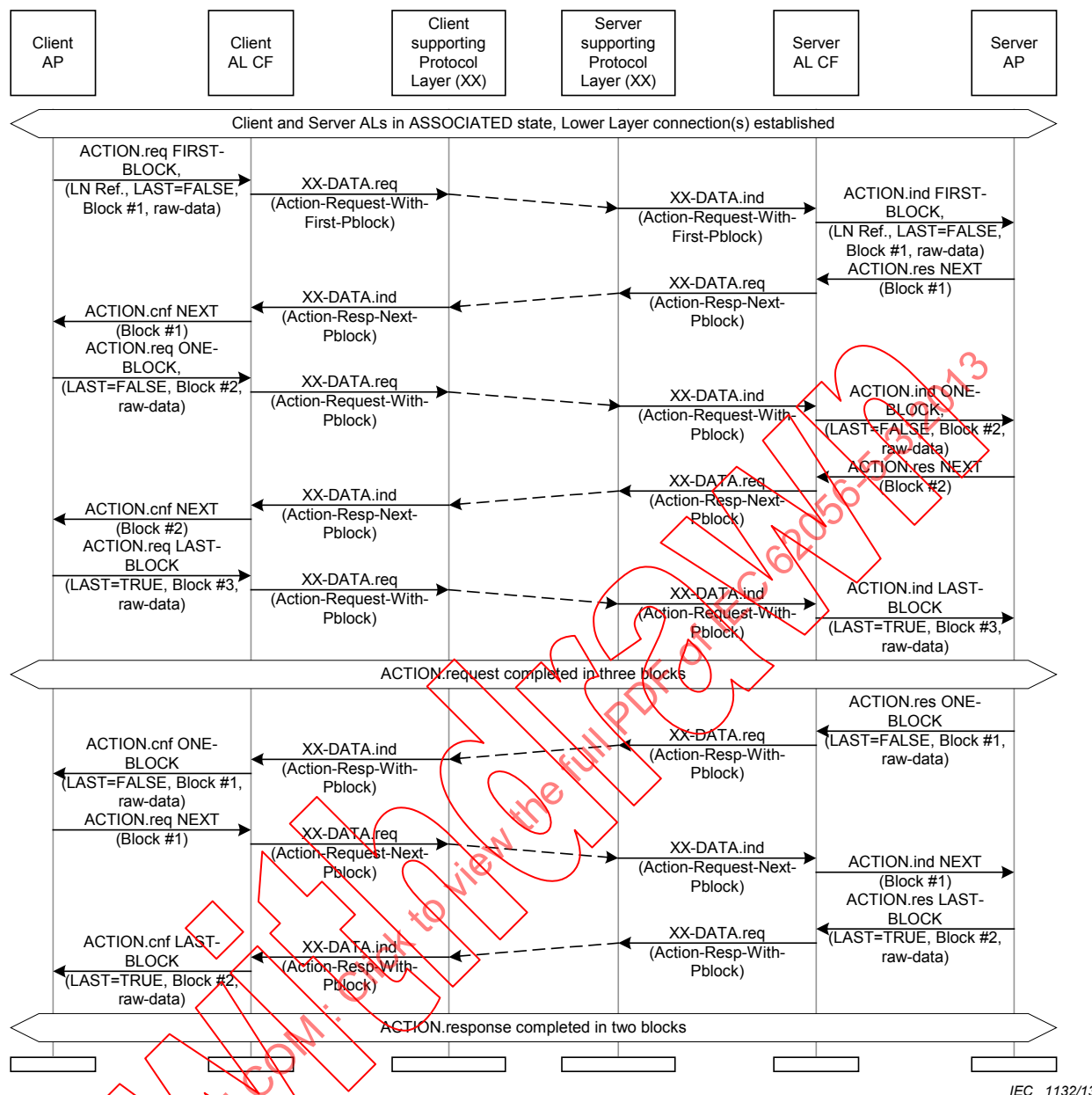
- lors de la première phase, le client envoie la primitive ACTION.request avec les paramètres d'appel de méthode des méthodes référencées, et le serveur en accuse réception. Le processus est essentiellement le même que dans le cas du service SET;
- lors de la seconde phase, le serveur envoie la primitive ACTION.response avec le résultat de l'appel de méthode et les paramètres de retour. Le processus est essentiellement le même que dans le cas du service GET;

Tout au long de la procédure, les paramètres Invoke_Id et Priority doivent être identiques dans chaque primitive.

Si lors d'un transfert de données long, le serveur reçoit une autre demande de service, celle-ci est traitée en fonction des règles de priorité et des paramètres de gestion des priorités (Bit 9 du bloc de conformité).

La Figure 21 montre le MSC correspondant au transfert par bloc dans les deux directions.

Si une erreur se produit lors d'un transfert de données long, le transfert doit être annulé. Les situations d'erreur sont identiques à celles des services GET et SET.



IEC 1132/13

Légende

Anglais	Français
Client AP	AP de client
Client AL CF	CF d'AL de client
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support de serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs in ASSOCIATED state, Lower Layer connection(s) established	Les AL de client et de serveur sont à l'état ASSOCIATED, la ou les connexions de la couche inférieure sont établies
ACTION.request completed in three blocks	ACTION.request transmise en trois blocs
ACTION.response completed in two blocks	ACTION.response transmise en deux blocs
LN Ref	Réf. LN

Figure 21 – MSC du service ACTION avec transfert de bloc

7.3.6 Protocole du service EventNotification

À l'appel du service EventNotification.request, l'AL de serveur génère une APDU EventNotificationRequest. Les possibilités d'envoi de cette APDU dépendent du profil de communication et du statut de connexion des couches inférieures. Par conséquent, le protocole du service EventNotification est présenté plus en détail dans les parties de la CEI 62056 qui décrivent les profils de communication;

Dans tous les cas, pour envoyer la/les valeur(s) de l'attribut/des attributs au client, sans que le client la/les demande:

- le serveur utilise la primitive de service EventNotification.request;
- à l'appel de cette primitive, l'AL de serveur génère une APDU EventNotificationRequest;
- cette APDU est transmise au client par le service de couche de support à la première occasion. Le type de service et la disponibilité de cette première occasion dépendent du profil de communication utilisé;
- à réception de l'APDU EventNotificationRequest, l'AL de client génère une primitive EventNotification.indication¹⁷ sur l'AP de client COSEM;
- par défaut, les notifications d'événement sont envoyées à partir du dispositif logique de gestion (serveur) vers l'AP de gestion (client).

7.3.7 Protocole du service Read

Comme expliqué en 6.12, le service Read est utilisé lorsque le serveur applique le référencement SN, soit pour lire un/des attribut(s) d'objet d'interface COSEM, soit pour appeler un/des méthode(s) lorsque des paramètres de retour sont attendus:

- dans le premier cas, les primitives de service GET.request sont mappées vers les primitives Read.request et les primitives Read.confirm sont mappées vers les primitives GET.confirm. Le mappage et les APDU SN correspondantes sont indiqués dans le Tableau 39;
- dans le second cas, les primitives de service ACTION.request sont mappées vers les primitives Read.request et les primitives Read.response sont mappées vers les primitives ACTION.response. Le mappage et les APDU SN correspondantes sont spécifiés dans le Tableau 40.

NOTE Dans les tables de mappage ci-dessous, la notation suivante est utilisée:

- pour les services LN, seuls les types de demande et de réponse sont indiqués sans paramètres de service;
- pour les services SN, le nom de la primitive de service est suivi par les paramètres de service, entre parenthèses. Les éléments de nom de paramètre de service sont en majuscules et sont associés par un trait de soulignement afin de ne constituer qu'une entité. Les paramètres pouvant être répétés sont entre accolades. Les choix possibles pour le paramètre Variable_Access_Specification sont répertoriés après le symbole "=". Les différents choix sont séparés par une barre verticale "|";
- pour les APDU SN, le nom de l'APDU est suivi du symbole "::=" et les champs sont entre parenthèses. Les éléments de nom de champ ne sont pas en majuscules et sont associés par un tiret afin de ne constituer qu'une seule entité. Les champs pouvant être répétés sont entre accolades. Les différents choix sont séparés par une barre verticale "|".

¹⁷ Côté client, il s'agit toujours de la primitive EventNotification.indication, indépendamment du schéma de référencement (LN ou SN) utilisé par le serveur.

Tableau 39 – Mappage entre le service GET et le service Read

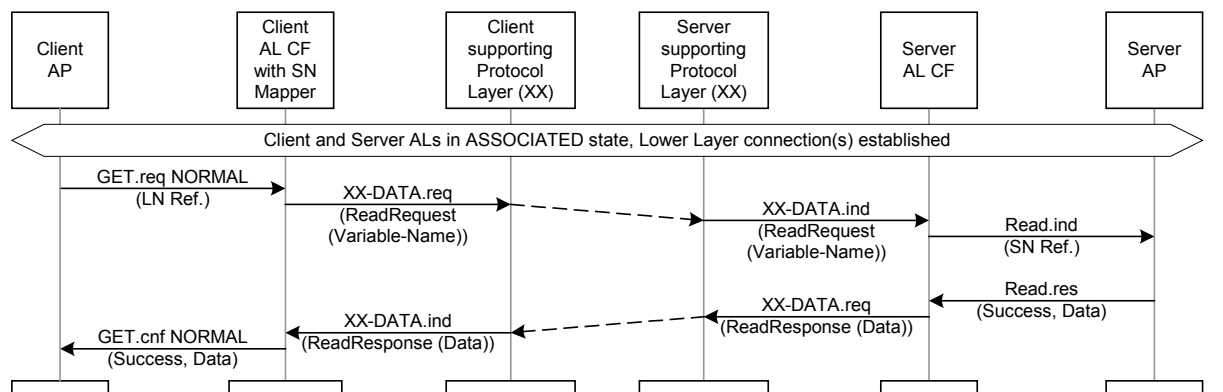
De la primitive GET.request de type	À la primitive Read.request	APDU SN
NORMAL	Read.request (Variable_Access_Specification) Variable_Access_Specification = Variable_Name Parameterized_Access;	ReadRequest::= (variable-name parameterized-access)
NEXT	Read.request (Variable_Access_Specification) Variable_Access_Specification = 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})
À la primitive GET.confirm de type	APDU SN	De la primitive 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) avec Last_Block = FALSE
LAST-BLOCK	ReadResponse::= (data-block-result)	Read.response (Data_Block_Result) avec Last_Block = TRUE
WITH-LIST	ReadResponse::= ({data data-access-error})	Read.response ({Data Data_Access_Error})

Tableau 40 – Mappage entre le service ACTION et le service Read

De la primitive ACTION.request de type	À la primitive Read.request	APDU SN
NORMAL	Read.request (Variable_Access_Specification) Variable_Access_Specification = Parameterized_Access; avec Variable_Name = référence de méthode, Selector = 0, Parameter = paramètre d'appel de méthode ou données NULL	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; avec Last_Block = FALSE, Block_Number = 1, Raw_Data = une partie des références de méthode et paramètre d'appel de méthode	ReadRequest::= (read-data-block-access)

De la primitive ACTION.request de type	À la primitive Read.request	APDU SN
ONE-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; avec Last_Block = FALSE, Block_Number = numéro suivant, Raw_Data = voir ci-dessus	ReadRequest::= (read-data-block-access)
LAST-BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; avec Last_Block = TRUE, Block_Number = numéro suivant, Raw_Data = voir ci-dessus	ReadRequest::= (read-data-block-access)
WITH-LIST	Read.request ({Variable_Access_Specification}) Variable_Access_Specification = Parameterized_Access; avec Variable_Name = référence de méthode, Selector = 0, Parameter = paramètre d'appel de méthode ou données NULL	ReadRequest::= ({{parameterized-access}})
WITH-LIST-AND-FIRST- BLOCK	Read.request (Variable_Access_Specification) Variable_Access_Specification = Read_Data_Block_Access; avec Last_Block = FALSE, Block_Number = 1, Raw_Data = voir ci-dessus	ReadRequest::= (read-data-block-access)
À la primitive ACTION.confirm	APDU SN	De la primitive 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; avec Last_Block = FALSE
LAST-BLOCK	ReadResponse::= (data-block-result)	Read.response (Read_Result) Read_Result = Data_Block_Result; avec 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;

La Figure 22 montre le MSC d'un service Read utilisé pour lire la valeur d'un attribut unique.



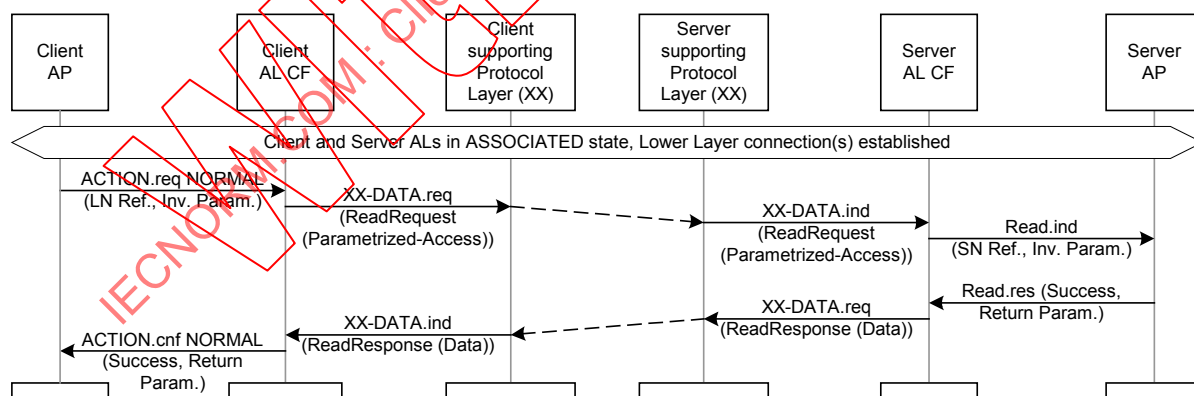
IEC 1133/13

Légende

Anglais	Français
Client AP	AP de client
Client AL CF with SN Mapper	CF d'AL de client avec mappeur SN
Client supporting Protocol Layer (XX)	Couche de protocole de support du client (XX)
Server supporting Protocol Layer (XX)	Couche de protocole de support de serveur (XX)
Server AL CF	CF d'AL de serveur
Server AP	AP de serveur
Client and Server ALs in ASSOCIATED state, Lower Layer connection(s) established	Les AL de client et de serveur sont à l'état ASSOCIATED, la ou les connexions de la couche inférieure sont établies
LN Ref	Réf. LN
SN Ref	Réf. SN

Figure 22 – MSC du service Read utilisé pour lire un attribut

La Figure 23 montre le MSC d'un service Read utilisé pour appeler une méthode unique.



IEC 1134/13

Légende: Idem Figure 15

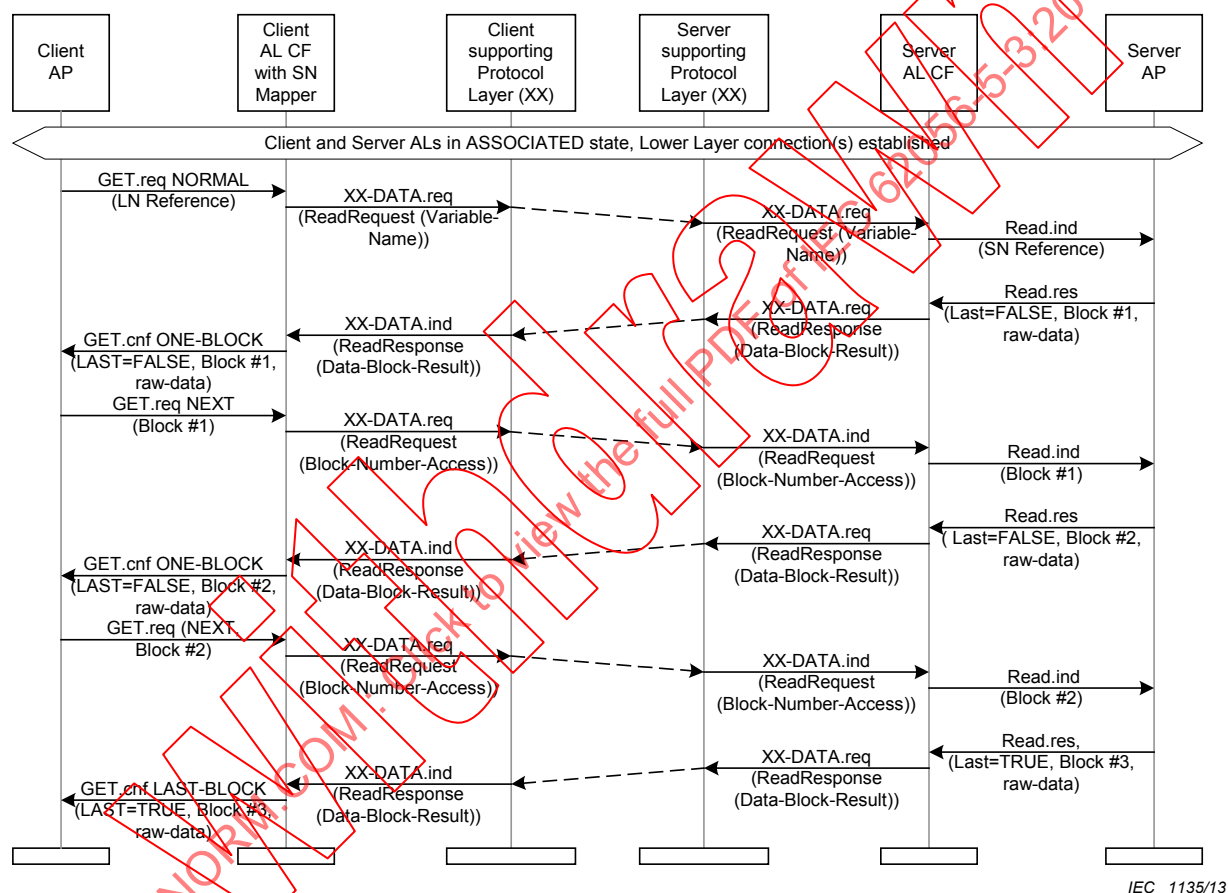
Figure 23 – MSC du service Read utilisé pour appeler une méthode

La Figure 24 indique le MSC du service Read utilisé pour lire un attribut unique, avec le retour du résultat en trois blocs.

Le processus de préparation et de transport de données longues est essentiellement le même que pour les services GET et ACTION.

- si le service Read est utilisé pour lire la valeur d'(un) attribut(s) d'objet COSEM, l'élément Raw_Data du résultat Data_Block_Result contient une partie de la liste des résultats Read_Result;
- si le service Read est utilisé pour appeler une/des méthode(s) d'objet COSEM et si de longs paramètres d'appel de méthodes sont à envoyer, l'élément Raw_Data du bloc Read_Data_Block_Access contient une partie de la/des référence(s) de méthode et du/des paramètre(s) d'appel de méthode. Si de longues réponses aux appels de méthode sont retournées, l'élément Raw_Data du bloc Data_Block_Result contient une partie de la/des réponse(s) d'appel de méthode.

Si une erreur se produit, il convient que le serveur retourne une primitive de service Read.response avec l'élément Data_Access_Error contenant les informations de diagnostic appropriées; par exemple, data-block-number-invalid.



IEC 1135/13

Légende: Idem Figure 22

Figure 24 – MSC du service Read utilisé pour lire un attribut, avec transfert de blocs

7.3.8 Protocole du service Write

Comme expliqué en 6.13, le service Write est utilisé lorsque le serveur utilise le référencement SN, soit pour écrire un/des attribut(s) d'objet COSEM, soit pour appeler une/des méthode(s) lorsqu'aucun paramètre de retour n'est attendu:

- dans le premier cas, les primitives de service SET.request sont mappées vers les primitives Write.request et les primitives Write.confirm sont mappées vers les primitives SET.confirm. Le mappage et les APDU SN correspondantes sont indiqués dans le Tableau 41;
- dans le second cas, les primitives de service ACTION.request sont mappées vers les primitives Write.request et les primitives Write.response vers les primitives

ACTION.confirm. Le mappage et les APDU SN correspondantes sont indiqués dans le Tableau 42.

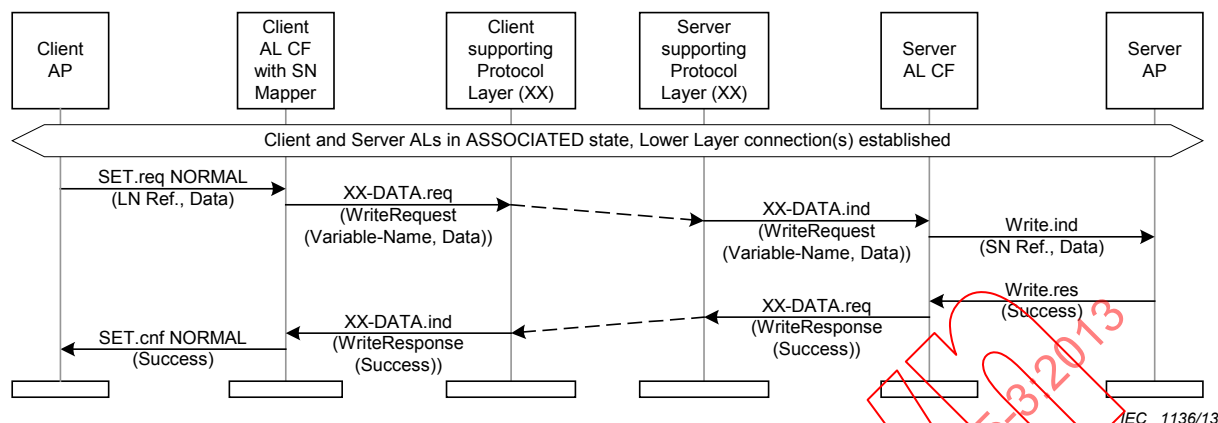
Tableau 41 – Mappage entre le service SET et le service Write

De la primitive SET.request de type	À la primitive Write.request	APDU SN
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; avec Last_Block = FALSE, Block_Number = 1, Data = raw-data, contenant la forme codée de la/des référence(s) d'attribut et les données d'écriture.	WriteRequest ::= (write-data-block-access, data)
ONE-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; avec Last_Block = FALSE, Block_Number = numéro suivant, Data = voir ci-dessus	WriteRequest ::= (write-data-block-access, data)
LAST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; avec Last_Block = TRUE, Block_Number = numéro suivant, Data = voir ci-dessus	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 = Write_Data_Block_Access; avec Last_Block = FALSE, Block_Number = 1 Data = voir ci-dessus	WriteRequest ::= (write-data-block-access, data)
À la primitive SET.confirm de type	APDU SN	De la primitive 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;

Tableau 42 – Mappage entre le service ACTION et le service Write

De la primitive ACTION.request de type	À la primitive Write.request	APDU SN
NORMAL	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name; Data = paramètres d'appel de méthode ou données NULL	WriteRequest::= (variable-name, data)
FIRST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; avec Last_Block = FALSE, Block_Number = 1, Data = raw-data, contenant la forme codée de la/des référence(s) de méthode et les paramètres d'appel de méthode;	WriteRequest::= (write-data-block-access, data)
ONE-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; avec Last_Block = FALSE, Block_Number = numéro suivant, Data = voir ci-dessus	WriteRequest::= (write-data-block-access, data)
LAST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; avec Last_Block = TRUE, Block_Number = numéro suivant, Data = voir ci-dessus	WriteRequest::= (write-data-block-access, data)
WITH-LIST	Write.request ({Variable_Access_Specification}, {Data}) Variable_Access_Specification = Variable_Name; Data = paramètres d'appel de méthode ou données NULL	WriteRequest::= ({variable-name}, {data})
WITH-LIST-AND-FIRST-BLOCK	Write.request (Variable_Access_Specification, Data) Variable_Access_Specification = Write_Data_Block_Access; avec Last_Block = FALSE, Block_Number = 1, Data = identique au premier bloc	WriteRequest::= (write-data-block-access, data)
À la primitive ACTION.confirm	APDU SN	De la primitive 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)
À la primitive ACTION.confirm	APDU SN	De la primitive Write.response
WITH-LIST	WriteResponse::= ({success data-access-error})	Write.response ({Write_Result}) Write_Result = Success Data_Access_Error;

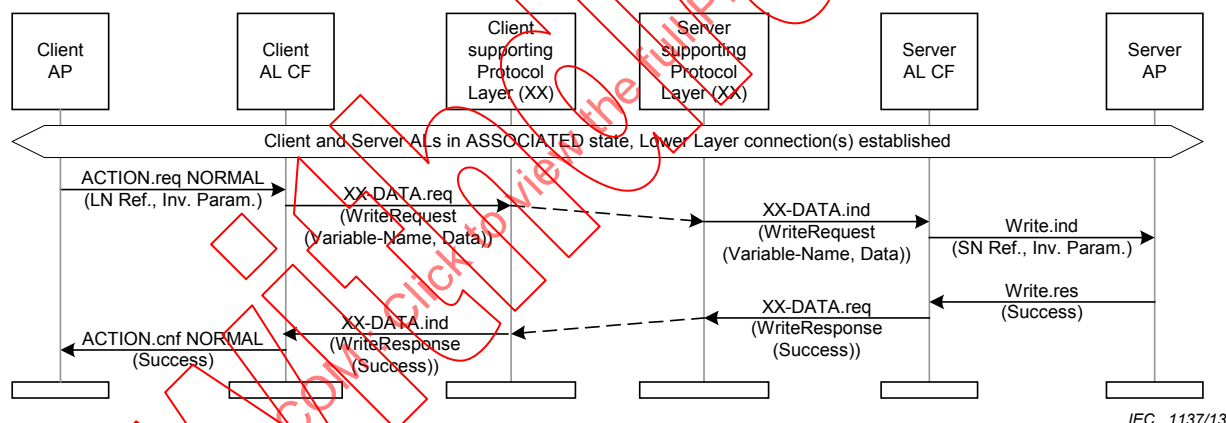
La Figure 25 montre le MSC d'un service Write utilisé pour écrire la valeur d'un seul attribut, en cas de succès.



Légende: Idem Figure 22

Figure 25 – MSC du service Write utilisé pour écrire un attribut

La Figure 26 montre le MSC d'un service Write utilisé pour appeler une méthode unique, en cas de succès.



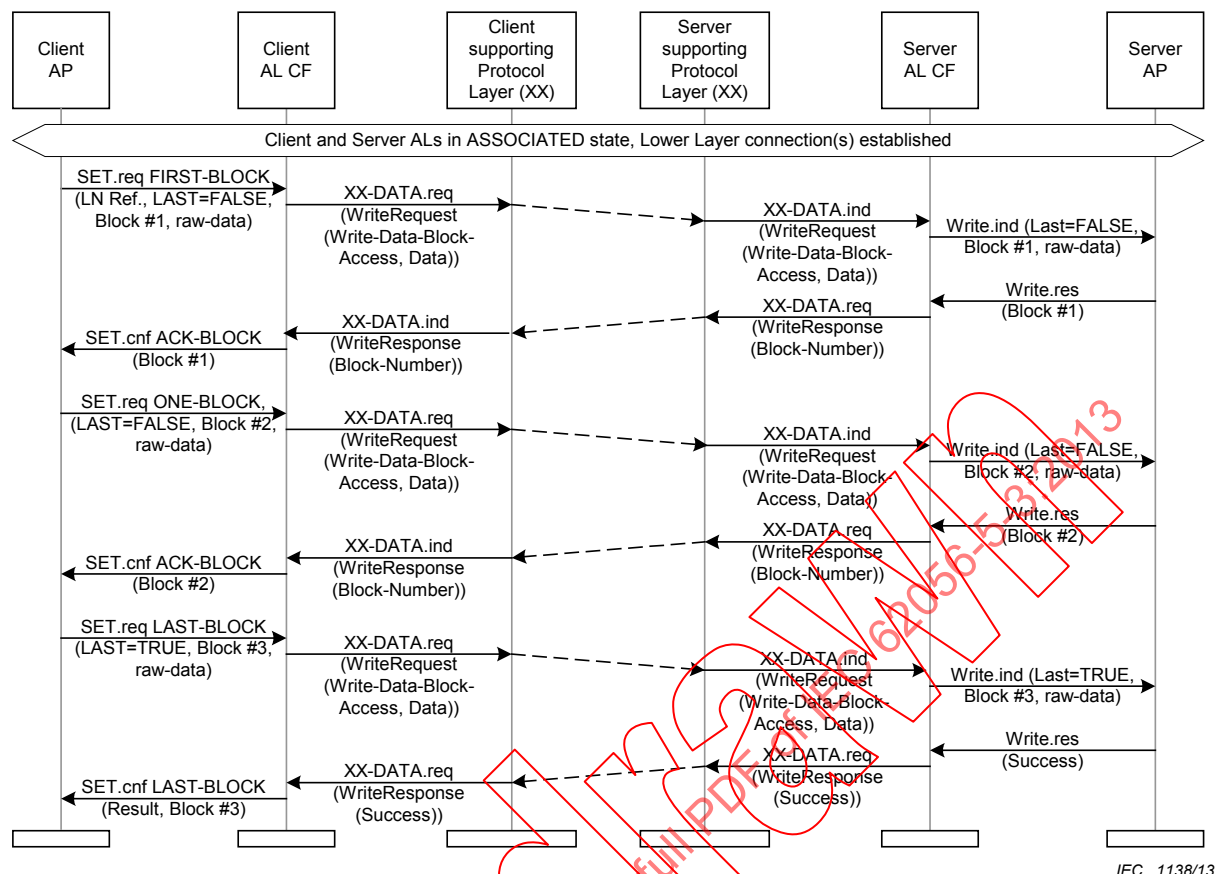
Légende: Idem Figure 15

Figure 26 – MSC du service Write utilisé pour appeler une méthode

La Figure 27 montre le MSC d'un service Write utilisé pour l'écriture d'un attribut unique, avec retour du résultat en trois blocs.

Le processus de préparation et de transport de données longues est essentiellement le même que pour les services SET et ACTION.

Si une erreur se produit, il convient que le serveur retourne une primitive de service Write.response avec l'élément Data_Access_Error contenant les informations de diagnostic appropriées; par exemple, data-block-number-invalid.



Légende: Idem Figure 15

Figure 27 – MSC du service Write utilisé pour écrire un attribut, avec transfert de blocs

7.3.9 Protocole du service UnconfirmedWrite

Ce service peut être appelé uniquement si une AA a déjà été établie. En fonction du profil de communication, l'APDU correspondant à la demande peut être transportée à l'aide des services de données sans connexion (CL) ou orientés sur la connexion (CO) de la couche de protocole de support.

Comme expliqué en 6.14, le service UnconfirmedWrite peut être utilisé soit pour écrire un/des attribut(s) d'objet COSEM, soit pour appeler une/des méthode(s) lorsqu'aucun paramètre de retour n'est attendu:

- dans le premier cas, les primitives de service SET.request sont mappées vers les primitives UnconfirmedWrite.request. Le mappage et les APDU SN correspondantes sont indiqués dans le Tableau 43;
- dans le deuxième cas, les primitives de service ACTION.request sont mappées vers les primitives UnconfirmedWrite.request. Le mappage et les APDU SN correspondantes sont indiqués dans le Tableau 44.

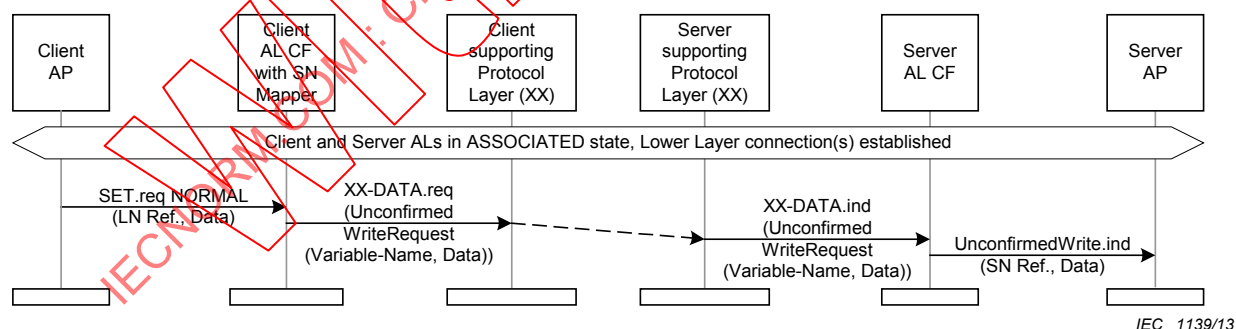
Tableau 43 – Mappage entre le service SET et le service UnconfirmedWrite

De la primitive SET.request de type	Vers la primitive UnconfirmedWrite.request	APDU SN
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})

Tableau 44 – Mappage entre le service ACTION et le service UnconfirmedWrite

De la primitive ACTION.request de type	À la primitive UnconfirmedWrite.request	APDU SN
NORMAL	UnconfirmedWrite.request (Variable_Access_Specification, Data) Variable_Access_Specification = Variable_Name; Data = paramètres d'appel de méthode ou données NULL	UnconfirmedWriteRequest::= (variable-name, data)
WITH-LIST	UnconfirmedWrite.request ({Variable_Access_Specification}, {Data}) Variable_Access_Specification = Variable_Name; Data = voir ci-dessus	UnconfirmedWriteRequest::= ({{variable-name}}, {data})

La Figure 28 montre le MSC d'un service Write utilisé pour écrire la valeur d'un seul attribut, en cas de succès.



Légende: Idem Figure 22

Figure 28 – MSC du service Unconfirmed Write utilisé pour écrire un attribut

7.3.10 Protocole du service InformationReport

Le protocole du service InformationReport, spécifié en 6.15, est essentiellement identique à celui du service EventNotification, voir 7.3.6.

Contrairement au service EventNotification, le service InformationReport ne contient pas le paramètre facultatif Application_Addresses. Le rapport d'informations est donc toujours envoyé par l'unité logique de gestion de serveur à l'AP de gestion du client.

À l'appel du service InformationReport.request, l'AP de serveur génère une APDU InformationReportRequest. Cette APDU est envoyée depuis le SAP du dispositif logique de gestion vers le SAP du dispositif gestion de client, à l'aide de services de données des couches inférieures, de façon non sollicitée, à la première occasion.

Les possibilités d'envoi de cette APDU dépendent du profil de communication et du statut de connexion des couches inférieures. Le protocole du service InformationReport est décrit plus en détails à l'Annexe A.

Le service InformationReport peut contenir plusieurs noms d'attributs et leur contenu. En revanche, le service EventNotification spécifié en 6.9 ne contient qu'un attribut de référence. Par conséquent, lorsque l'APDU InformationReportRequest contient plusieurs attributs, elle doit être mappée vers plusieurs services EventNotification.ind, comme indiqué dans le Tableau 45.

Tableau 45 – Mappage entre les services EventNotification et InformationReport

EventNotification.ind (un ou plusieurs)	InformationReport.ind
Time (facultatif)	Current-time (facultatif)
COSEM_Class_Id, COSEM_Object_Instance_Id, COSEM_Object_Attribute_Id	Variable_Name (Variable_Name)
Attribute_Value	Data {Data}

8 Syntaxe abstraite des APDU ACSE et COSEM

La syntaxe abstraite des APDU COSEM est spécifiée dans le présent Article 8 à l'aide d'ASN.1. Voir l'ISO/CEI 8824-1.

COSEMpdu 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
}
```

-- Les étiquettes CI-PDU sont supérieures à 24, chiffre qui correspond à la dernière APDU DLMS déchiffrée spécifiée dans la norme
-- CEI 61334-4-41.

}

xDLMS-APDU ::= CHOICE

```
{
    -- PDU xDLMS normalisées utilisées dans DLMS/COSEM
    -- PDU DLMS normalisées
}
```

-- sans chiffrement

```
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]

unconfirmedWriteRequest   [22] IMPLICIT  UnconfirmedWriteRequest,
informationReportRequest  [24] IMPLICIT  InformationReportRequest,
```

```
-- L'étiquette APDU de chaque APDU xDLMS chiffrée indique le type de chaque APDU non
chiffrée
-- et si la clé utilisée est globale ou dédiée. Le type de clé est contenu dans
-- l'en-tête de sécurité, et après avoir supprimé le codage et/ou vérifié
-- l'étiquette d'authentification, l'APDU d'origine et son étiquette sont restaurées.
Par conséquent les
-- étiquettes des APDU chiffrées contiennent des informations redondantes, mais elles
sont conservées
-- à des fins de cohérence.

-- avec chiffrage global

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,

-- avec chiffrage dédié

-- non utilisé dans DLMS/COSEM
ded-initiateRequest      [65]  IMPLICIT  OCTET STRING,
ded-readRequest          [69]  IMPLICIT  OCTET STRING,
ded-writeRequest         [70]  IMPLICIT  OCTET STRING,

-- non utilisé dans 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,

-- les quatre APDU ACSE
aarq                     AARQ-apdu,
aare                     AARE-apdu,
rlrq                     RLREQ-apdu,          -- FACULTATIF
rlre                     RLRE-apdu           -- FACULTATIF

-- APDU xDLMS utilisées avec le référencement LN
-- sans chiffrage

get-request              [192]  IMPLICIT  GET-Request,
set-request              [193]  IMPLICIT  SET-Request,
event-notification-request [194]  IMPLICIT  EVENT-NOTIFICATION-Request,
action-request           [195]  IMPLICIT  ACTION-Request,
get-response              [196]  IMPLICIT  GET-Response,
set-response              [197]  IMPLICIT  SET-Response,
action-response           [199]  IMPLICIT  ACTION-Response,

-- avec chiffrage global

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,

-- avec chiffrage dédié

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
```

-- PDU de réponse d'exception

```

exception-response          [216] IMPLICIT ExceptionResponse

-- reserved                 [230] -- 0xE6
-- reserved                 [231] -- 0xE7
}

```

AARQ-apdu ::= [APPLICATION 0] **IMPLICIT SEQUENCE**

```

{
-- [APPLICATION 0] == [ 60H ] = [ 96 ]

protocol-version            [0] IMPLICIT BIT STRING {version1 (0)}
                             DEFAULT {version1},
application-context-name    [1] Application-context-name,
called-AP-title             [2] AP-title OPTIONAL,
called-AE-qualifier         [3] AE-qualifier OPTIONAL,
called-AP-invocation-id     [4] AP-invocation-identifier OPTIONAL,
called-AE-invocation-id     [5] AE-invocation-identifier OPTIONAL,
calling-AP-title            [6] AP-title OPTIONAL,
calling-AE-qualifier        [7] AE-qualifier OPTIONAL,
calling-AP-invocation-id    [8] AP-invocation-identifier OPTIONAL,
calling-AE-invocation-id    [9] AE-invocation-identifier OPTIONAL,

-- Le champ suivant ne doit pas être présent si seul le kernel est utilisé.
sender-acse-requirements    [10] IMPLICIT ACSE-requirements OPTIONAL,

-- Le champ suivant doit uniquement être présent si l'unité fonctionnelle
d'authentification est
-- sélectionnée.
mechanism-name              [11] IMPLICIT Mechanism-name OPTIONAL,

-- Le champ suivant doit uniquement être présent si l'unité fonctionnelle
d'authentification est
-- sélectionnée.
calling-authentication-value [12] EXPLICIT Authentication-value OPTIONAL,
implementation-information  [29] IMPLICIT Implementation-data OPTIONAL,
user-information            [30] EXPLICIT Association-information OPTIONAL
}

-- Le champ user-information doit transmettre une APDU InitiateRequest xDLMS codée en
A-XDR.
-- L'OCTET STRING résultant doit être codé en 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,

-- Le champ suivant ne doit pas être présent si seul le Kernel est utilisé.
responder-acse-requirements [8] IMPLICIT ACSE-requirements OPTIONAL,

-- Le champ suivant doit uniquement être présent si l'unité fonctionnelle
d'authentification est
-- sélectionnée.
mechanism-name              [9] IMPLICIT Mechanism-name OPTIONAL,

-- Le champ suivant doit uniquement être présent si l'unité fonctionnelle
d'authentification est
-- sélectionnée.
responding-authentication-value [10] EXPLICIT Authentication-value OPTIONAL,
implementation-information      [29] IMPLICIT Implementation-data OPTIONAL,
user-information                [30] EXPLICIT Association-information OPTIONAL
}

-- Le champ user-information doit transmettre une APDU InitiateResponse (ou, si le
-- contexte xDLMS proposé est refusé par le serveur, une APDU ConfirmedServiceError)

```

-- codée en A-XDR. L'OCTET STRING résultant doit être codé en 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
}
```

-- Le champ user-information de l'APDU RLRQ / RLRE peut contenir une APDU InitiateRequest

-- codée en A-XDR, puis coder l'OCTET STRING résultant en BER, lorsque l'AA à libérer utilise le chiffrement.

-- types utilisés dans les champs des APDU ACSE, selon leur ordre d'apparition

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,
    external                                  [2] IMPLICIT    EXTERNAL,
    other                                     [3] IMPLICIT    SEQUENCE
    {
        other-mechanism-name                 Mechanism-name,
        other-mechanism-value                ANY DEFINED BY other-mechanism-name
    }
}
```

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),
        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),

```



```

reserved-six (6),
reserved-seven (7),
attribute0-supported-with-set (8),
priority-mgmt-supported (9),
attribute0-supported-with-get (10),
block-transfer-with-get-or-read (11),
block-transfer-with-set-or-write (12),
block-transfer-with-action (13),
multiple-references (14),
information-report (15),
reserved-sixteen (16),
reserved-seventeen (17),
parameterized-access (18),
get (19),
set (20),
selective-access (21),
event-notification (22),
action (23)
} (SIZE(24))
ObjectName ::= Integer16
-- pour les objets de variable désignés (noms courts), les trois derniers bits doivent
être mis à 000;
-- pour les objets vaa-name, les trois derniers bits doivent être mis à 111.

-- L'APDU Confirmed ServiceError est utilisée uniquement avec les APDU InitiateRequest,
ReadRequest
--et WriteRequest lorsque la demande échoue, afin de fournir des informations de
diagnostic.

ConfirmedServiceError ::= CHOICE
{
-- l'étiquette 0 est réservée
-- Dans DLMS/COSEM, seuls initiateError, Read et Write sont pertinents

initiateError [1] ServiceError,
getStatus [2] ServiceError,
getNameList [3] ServiceError,
getVariableAttribute [4] ServiceError,
read [5] ServiceError,
write [6] ServiceError,
getDataSetAttribute [7] ServiceError,
getTIAttribute [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
{
-- Fournisseur DLMS uniquement
other (0),
time-elapsed (1), -- temps écoulé depuis l'envoi de la demande
application-unreachable (2), -- AEi pair non joignable
application-reference-invalid (3), -- problème lié à l'adressage
application-context-unsupported (4), -- incompatibilité du contexte d'application
provider-communication-error (5), -- erreur sur l'équipement local ou distant
deciphering-error (6), -- erreur détectée par la fonction de
déchiffage
},

hardware-resource [1] IMPLICIT ENUMERATED
{
-- problèmes liés au matériel VDE
other (0),
memory-unavailable (1),
processor-resource-unavailable (2),
mass-storage-unavailable (3),

```



```

    other-resource-unavailable          (4)
},

vde-state-error                        [2] IMPLICIT ENUMERATED
{
    -- Description de la source d'erreur
    other (0),
    no-dlms-context                    (1),
    loading-data-set                   (2),
    status-nochange                     (3),
    status-inoperable                   (4)
},

service                               [3] IMPLICIT ENUMERATED
{
    -- problèmes liés à la gestion de services
    other (0),
    pdu-size                           (1), -- pdu trop longue
    service-unsupported                 (2) -- tel que défini dans le bloc de conformité
},

definition                            [4] IMPLICIT ENUMERATED
{
    -- problèmes liés aux objets dans un service
    other (0),
    object-undefined                   (1), -- objet non défini dans le VDE
    object-class-inconsistent          (2), -- classe d'objet incompatible avec le
service
    object-attribute-inconsistent      (3) -- les attributs d'objets ne sont pas
cohérents
},

access                                [5] IMPLICIT ENUMERATED
{
    -- erreur lors de l'accès à l'objet
    other (0),
    scope-of-access-violated           (1), -- accès refusé par la raison d'autorisation
    object-access-violated             (2), -- accès incompatible avec l'attribut d'objet
    hardware-fault                     (3), -- échec d'accès pour raison matérielle
    object-unavailable                 (4) -- le VDE tient l'objet pour non disponible
},

initiate                              [6] IMPLICIT ENUMERATED
{
    -- lancer erreur de service
    other (0),
    dlms-version-too-low               (1), -- la version de DLMS proposée est trop
faible
    incompatible-conformance           (2), -- le service proposé n'est pas suffisant
    pdu-size-too-short                 (3), -- la taille de PDU proposée est trop faible
    refused-by-the-VDE-Handler         (4) -- création de vaa impossible ou non
autorisée
},

load-data-set                          [7] IMPLICIT ENUMERATED
{
    -- erreur des services de chargement de jeu de données
    other (0),
    primitive-out-of-sequence          (1), -- en fonction des transitions de l'état de
chargement de DataSet
    not-loadable                       (2), -- attribut chargeable défini sur FALSE
    dataset-size-too-large             (3), -- taille évaluée de Data Set trop importante
    not-awaited-segment                (4), -- segment proposé non attendu
    interpretation-failure             (5), -- erreur d'interprétation de segment
    storage-failure                    (6), -- erreur de stockage de segment
    data-set-not-ready                 (7) -- État de Data Set incorrect pour
-- le téléchargement
},

-- change-scope                       [8] IMPLICIT ENUMERATED

task                                  [9] IMPLICIT ENUMERATED
{
    -- Erreur de services TI

```