Electricity metering — Data exchange for meter reading, tariff and load control —

Part 53: COSEM application layer

The European Standard EN 62056-53:2007 has the status of a British Standard

ICS 35.100.70; 91.140.50

British Standards

National foreword

This British Standard was published by BSI. It is the UK implementation of EN 62056-53:2007. It is identical with IEC 62056-53:2006. It supersedes BS EN 62056-53:2002, which will be withdrawn on 1 February 2010.

The UK participation in its preparation was entrusted to Technical Committee PEL/13, Electricity meters.

A list of organizations represented on this committee can be obtained on request to its secretary.

This publication does not purport to include all the necessary provisions of a contract. Users are responsible for its correct application.

Compliance with a British Standard cannot confer immunity from legal obligations.

This British Standard was published under the authority of the Standards Policy and Strategy Committee on 30 June 2007

ISBN 978 0 580 50854 7

Amendments issued since publication

Amd. No.	Date	Comments

EUROPEAN STANDARD

NORME EUROPÉENNE

EUROPÄISCHE NORM

EN 62056-53

April 2007

ICS 91.140.50; 35.100.70

Supersedes EN 62056-53:2002

English version

Electricity metering - Data exchange for meter reading, tariff and load control - Part 53: COSEM application layer

(IEC 62056-53:2006)

Equipements de mesure
de l'énergie électrique -
Echange des données
pour la lecture des compteurs,
le contrôle des tarifs et de la charge -
Partie 53: Couche application COSEM
(CEI 62056-53:2006)

Messung der elektrischen Energie -
Zählerstandsübertragung,
Tarif- und Laststeuerung -
Teil 53: COSEM-Anwendungsschicht
(IEC 62056-53:2006)

This European Standard was approved by CENELEC on 2007-02-01. CENELEC members are bound to comply with the CEN/CENELEC Internal Regulations which stipulate the conditions for giving this European Standard the status of a national standard without any alteration.

Up-to-date lists and bibliographical references concerning such national standards may be obtained on application to the Central Secretariat or to any CENELEC member.

This European Standard exists in three official versions (English, French, German). A version in any other language made by translation under the responsibility of a CENELEC member into its own language and notified to the Central Secretariat has the same status as the official versions.

CENELEC members are the national electrotechnical committees of Austria, Belgium, Bulgaria, Cyprus, the Czech Republic, Denmark, Estonia, Finland, France, Germany, Greece, Hungary, Iceland, Ireland, Italy, Latvia, Lithuania, Luxembourg, Malta, the Netherlands, Norway, Poland, Portugal, Romania, Slovakia, Slovenia, Spain, Sweden, Switzerland and the United Kingdom.

CENELEC
European Committee for Electrotechnical Standardization
Comité Européen de Normalisation Electrotechnique
Europäisches Komitee für Elektrotechnische Normung

Central Secretariat: rue de Stassart 35, B - 1050 Brussels

Foreword

The text of document 13/1387/FDIS, future edition 2 of IEC 62056-53, prepared by IEC TC 13, Electrical energy measurement, tariff- and load control, was submitted to the IEC-CENELEC parallel vote and was approved by CENELEC as EN 62056-53 on 2007-02-01.

This European Standard supersedes EN 62056-53:2002.

The main changes with respect to EN 62056-53:2002 are as follows:

– the protocol of the COSEM-RELEASE service has been changed: depending on the communication profile used, these services may rely on the ACSE A_RELEASE services;

– the parsing order of the AARQ APDU has been changed;

– handling of repeated application association requests has been simplified;

– the Service_Class parameter of the COSEM-OPEN service is now linked to the response allowed field of the xDLMS-Initiate.request APDU;

– the Service_Class parameter of COSEM services for data exchange using LN referencing is now linked to bit 6 of the Invoke-Id-And-Priority parameter;

– a new, optional EXCEPTION APDU has been introduced. The server may send back this APDU after an erroneous service request;

– a general part about using the COSEM application layer in various communication profiles has been added;

– the description of using the COSEM Application layer in the 3-layer, connection-oriented, HDLC based communication profile has been amended;

– a new, TCP-UDP/IP based communication profile has been defined.

The following dates were fixed:

– latest date by which the EN has to be implemented at national level by publication of an identical national standard or by endorsement	(dop)	2007-11-01
– latest date by which the national standards conflicting with the EN have to be withdrawn	(dow)	2010-02-01

The International Electrotechnical Commission (IEC) and CENELEC draw attention to the fact that it is claimed that compliance with this International Standard / European Standard may involve the use of a maintenance service concerning the stack of protocols on which the present standard IEC 62056-53 / EN 62056-53 is based.

The IEC and CENELEC take 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 with 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:

DLMS [1] User Association
Geneva / Switzerland
www.dlms.ch

[1] Device Language Message Specification

Annex ZA has been added by CENELEC.

Endorsement notice

The text of the International Standard IEC 62056-53:2006 was approved by CENELEC as a European Standard without any modification.

CONTENTS

ELECTRICITY METERING – DATA EXCHANGE FOR METER READING, TARIFF AND LOAD CONTROL –

Part 53: COSEM application layer

1 Scope

This part of IEC 62056 specifies the COSEM application layer in terms of structure, services and protocols for COSEM clients and servers, and defines how to use the 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 62056-62, using either logical name (LN) or short name (SN) referencing.

Annex A describes the xDLMS application service element.

Annex B defines how to use the COSEM application layer in various communication profiles.

Annex C includes encoding examples for APDUs.

Annex D gives an explanation of the role of data models and protocols in electricity meter data exchange.

2 Normative references

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

IEC 60050-300:2001, *International Electrotechnical Vocabulary (IEV) – 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 61334-4-41:1996, *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 62051:1999, *Electricity metering – Glossary of terms*

IEC 62051-1:2004, *Electricity metering – Data exchange for meter reading, tariff and load control – Glossary of Terms – Part 1: Terms related to data exchange with metering equipment using DLMS/COSEM*

IEC 62056-21:2002, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 21: Direct local data exchange*

IEC 62056-42:2002, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 42: Physical layer services and procedures for connection-oriented asynchronous data exchange*

IEC 62056-46:2002, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 46: Data link layer using HDLC protocol*
Amendment 1[2]

IEC 62056-47, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 47: COSEM transport layer for IP networks*

IEC 62056-61, Ed.2, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 61: Object identification system (OBIS)*

IEC 62056-62, Ed.2, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 62: Interface classes*

ISO/IEC 8649:1996, *Information technology – Open Systems Interconnection – Service definition for the Association Control Service Element*

ISO/IEC 8650-1:1996, *Information technology – Open systems interconnection – Connection-oriented protocol for the Association Control Service Element: Protocol specification*

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

ISO/IEC 8825*, Information technology – ASN.1 encoding rules*

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

STD0005 – *Internet Protocol*
Author: J. Postel
Date: September 1981
Also: RFC0791, RFC0792, RFC0919, RFC0922, RFC0950, RFC1112

STD0006 – *User Datagram Protocol*
Author: J. Postel
Date: 28 August 1980
Also: RFC0768

STD0007 – *Transmission Control Protocol*
Author: J. Postel
Date: September 1981
Also: RFC0793

See also Bibliography for other related Internet RFCs.

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purposes of this part of IEC 62056, the definitions in IEC 60050-300, IEC 62051 and IEC 62051-1 apply.

3.2 Abbreviations

.cnf	.confirm service primitive
.ind	.indication service primitive

[2] To be published.

.req	.request service primitive
.res	.response service primitive
AA	Application Association
AARE	Application Association REsponse
AARQ	Application Association ReQuest
ACSE	Application Control Service Element
AE	Application Entity
AP	Application Process
APDU	Application layer Protocol Data Unit
API	Application Programming Interface
ARP	Address Resolution Protocol
ASE	Application Service Element
ASO	Application Service Object
ATM	Asynchronous Transfer Mode
A-XDR	Adapted eXtended Data Representation
BER	Basic Encoding Rules
CF	Control function
CO	Connection Oriented
COSEM	COmpanion Specification for Energy Metering
DLMS	Device Language Message Specification
DSAP	Data link Service Access Point
FDDI	Fibre Distributed Data Interface
FTP	File Transfer Protocol
GMT	Greenwich Mean Time
GSM	Global System for Mobile communications
HDLC	High-level Data Link Control
HLS	High Level Security
HTTP	Hypertext Transfer Protocol
IC	Interface Class
IETF	Internet Engineering Task Force
IP	Internet Protocol
LAN	Local Area Network
LLC	Logical Link Control (sub-layer)
LLS	Low Level Security
LPDU	LLC Protocol Data Unit
LSAP	LLC sub-layer Service Access Point
LSB	Least Significant Bit
MAC	Medium Access Control
MD5	Message Digest Algorithm 5
MIB	Management Information Base
MSB	Most Significant Bit
MSC	Message Sequence Chart
OBIS	OBject Identification System

OSI	Open System Interconnection
PDU	Protocol Data Unit
PPP	Point-to-Point Protocol
PSTN	Public Switched Telephone Network
RARP	Reverse Address Resolution Protocol
RFC	Request For Comment
RLRQ	Release Request
RLRE	Release Response
SAP	Service Access Point
SHA-1	Secure Hash Algorithm 1
SNMP	Simple Network Management Protocol
VAA	Virtual Application Association
xDLMS-ASE	extended DLMS Application Service Element

4 The COSEM communications framework

4.1 Client/server type operation, communication profiles

Communication with electricity metering equipment using the COSEM interface object model is based on the **client/server** paradigm where metering equipment[3] plays the server role. In this environment, communication takes place always between a client and a server AP: in other words, the server AP provides remote services to the client AP. These services are provided via exchanging messages (SERVICE.requests/.responses) between the client and the server APs, as shown in Figure 1.

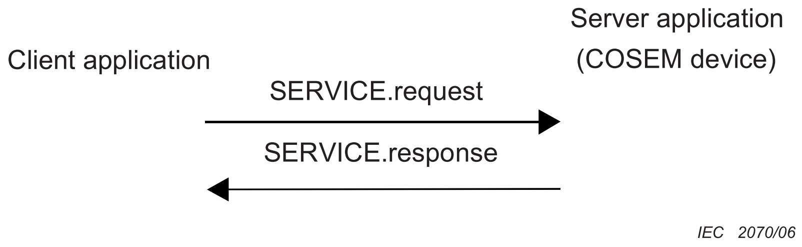


Figure 1 – Client/server relationship in COSEM

In general, the client and the server APs are located in separate devices; exchanging messages is done with the help of the communication protocol as shown in Figure 2.

[3] The term metering equipment is an abstraction; consequently the equipment playing the role of a server may be any type of equipment for which this abstraction is suitable.

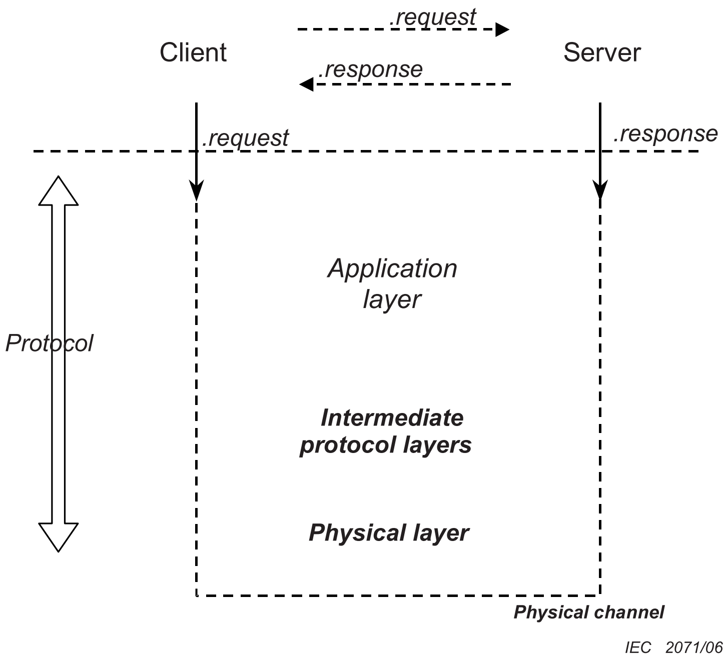


Figure 2 – Exchanging messages via the communication protocol

In general, communication protocols are structured in layers. The client and server COSEM applications use services of the highest protocol layer, that of the application layer: consequently, this is the only protocol layer containing COSEM specific element(s). This is called the xDLMS_ASE. All COSEM interface object related services – the xDLMS application protocol – are provided by this xDLMS_ASE.

Other protocol layers are independent of the COSEM model. Consequently, the COSEM application layer can be placed on the top of a wide variety of lower protocol layer stacks, as shown in Figure 3.

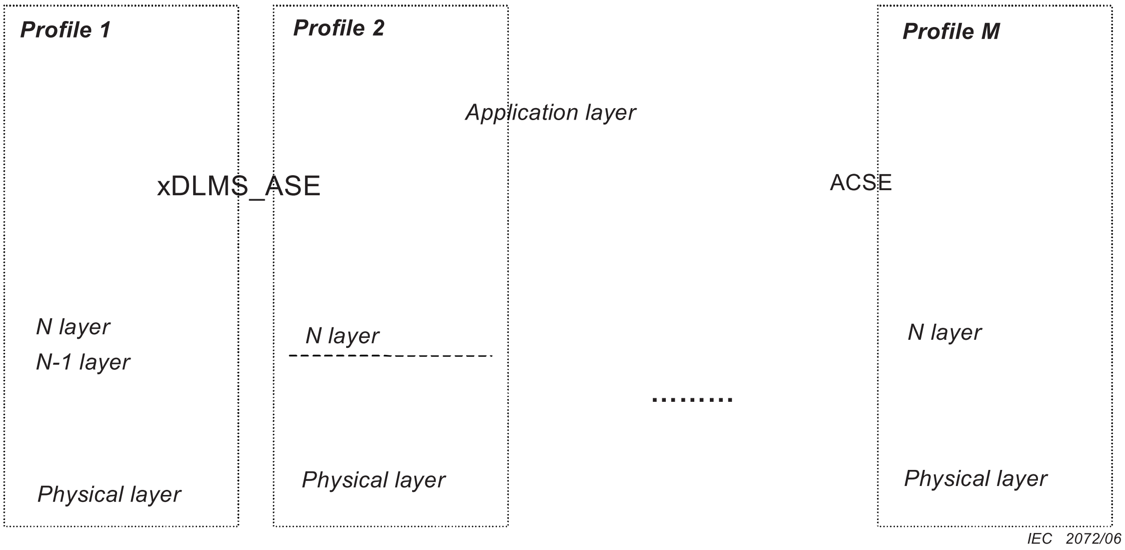


Figure 3 – The COSEM application layer on the top of various lower layer stacks

A complete protocol stack – including the application layer, a physical layer and all protocol layers between these extreme layers – is called a communication profile.

A communication profile is characterized by the protocol layers included, their parameters, and by the type – connection-oriented or connectionless – of the ACSE[4] included in the application layer.

4.2 Connection (association) oriented operation

The xDLMS application protocol is a connection-oriented protocol. It means that the client and server APs can use the services of the xDLMS_ASE only when these APs are associated[5]. Therefore, in this environment a communication session consists of three phases, as shown in Figure 4.

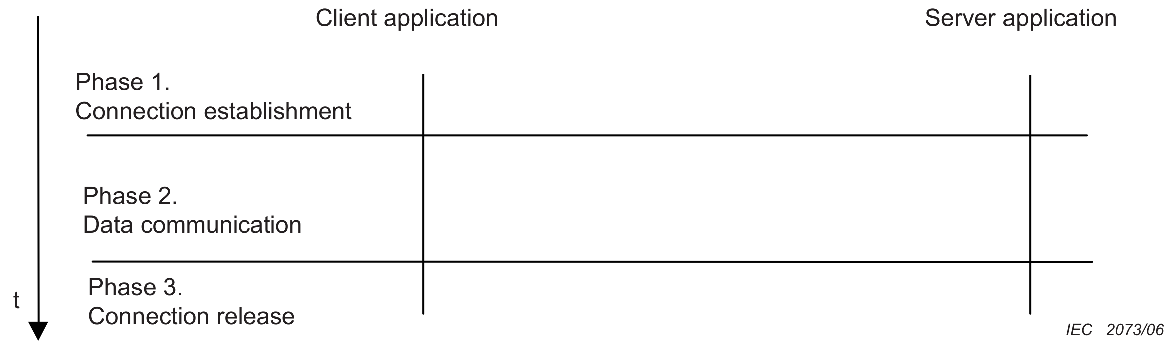


Figure 4 – A complete communication session in the CO environment

In the DLMS/COSEM environment, application association establishment is normally done by using the association request/response services of the standard association control service element. On the other hand, for the purposes of very simple devices, one-way communicating devices and for multicasting and broadcasting, pre-established application associations are also allowed; see 6.3.3. For such associations, there is no need to use the services of the ACSE: a full communication session may include only the data communication phase. (It can be considered that the connection establishment phase has been already done somewhere in the past.)

5 Overview: the COSEM application layer

5.1 Specification method

The COSEM application layer is specified in terms of *structure, services,* and *protocols*.

5.2 Application layer structure

The main component of the client and server COSEM application layers is the COSEM ASO, which provides services to the COSEM AP, and uses services provided by the supporting lower layer.

Both the client and server side COSEM ASO contain three mandatory components:

- the ACSE. The task of this element is to establish, maintain, and release application associations. For the purposes of connection-oriented profiles, the connection-oriented ACSE, specified in ISO/IEC 8649 and ISO/IEC 8650-1 is used;

[4] ACSE = Association Control Service Element

[5] Application associations can be considered as application level connections.

- the Extended DLMS application service element (xDLMS_ASE). The task of this element is to provide data communication services between COSEM APs. See also Annex A;
- the Control function (CF). This element specifies how the ASO services invoke the appropriate service primitives of the ACSE and the xDLMS ASE and the services of the supporting layer.

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

Figure 5 shows ‘minimal’ COSEM ASOs, containing only the three mandatory components.

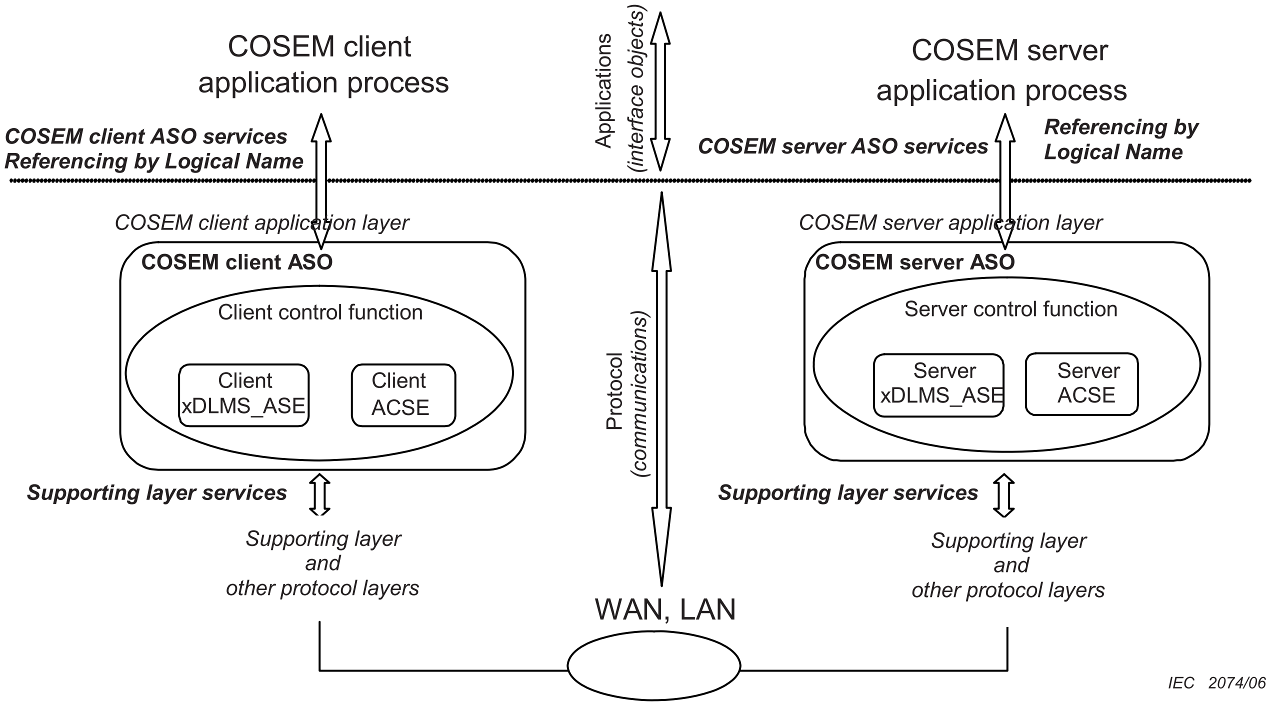

COSEM client application process
COSEM client ASO services
Referencing by Logical Name
Applications (interface objects)
COSEM server application process
COSEM server ASO services
Referencing by Logical Name
COSEM client application layer
COSEM client ASO
Client control function
Client xDLMS_ASE
Client ACSE
COSEM server application layer
COSEM server ASO
Server control function
Server xDLMS_ASE
Server ACSE
Protocol (communications)
Supporting layer services
Supporting layer and other protocol layers
Supporting layer services
Supporting layer and other protocol layers
WAN, LAN
IEC 2074/06

Figure 5 – The structure of the COSEM application layers

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

- for encoding the ACSE APDUS- AARQ, AARE, RLRQ, RLRE – BER encoding is used (ISO/IEC 8825);
- for encoding the APDUs carrying the data communication services, A-XDR encoding is used (IEC 61334-6).

5.3 Service specification

Service specifications cover the services required of, or by the COSEM client and server APs at the logical interfaces with the respective COSEM application layer, using connection-oriented procedures.

Services provided by the COSEM ASO fall into three categories:

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

The client and server application layer services are specified in Clause 6.

5.3.1 Services provided for application association establishment and release

These services are the following:

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

The COSEM-OPEN service is used during AA establishment phase and relies on the association request/response services of the ACSE. In the case of pre-established application associations (6.3.3) these services are not used.

In certain COSEM communication profiles – for example in the 3-layer, connection-oriented, HDLC based profile – there is a one-to-one relationship between a confirmed AA and the supporting protocol layer connection. In this case, the COSEM-RELEASE service used during the association release phase does not rely on the ACSE A_RELEASE services. Confirmed AAs in these profiles are released simply by disconnecting the corresponding lower layer connection.

Optionally, the COSEM-RELEASE service may rely on the ACSE A_RELEASE service. In some communication profiles, like in the TCP-UDP/IP based profile, using the ACSE A_RELEASE services for releasing COSEM AAs is mandatory.

5.3.2 Data communication services

IEC 62056-62 specifies two referencing methods for COSEM servers: referencing by Logical Names (LN) and referencing by Short Names (SN). Therefore, two distinct service sets are specified for the server side xDLMS_ASE. One set uses exclusively LN references the other set uses exclusively SN references. Thus, these services are the following:

- COSEM interface object attribute-related services: GET, SET for LN referencing and Read, Write, Unconfirmed Write for SN referencing;
- COSEM interface object method-related services: ACTION (LN), Read, Write or UnconfirmedWrite (SN);
- the EventNotification (LN), InformationReport (SN) services.

The services rely on the services of the xDLMS_ASE. Most of these services contain references to attributes or methods of COSEM interface objects.

The service set to be used on the server side during the data communications phase is negotiated during the association establishment phase using the conformance block, see 8.5. It shall not change during the lifetime of the established association. Using LN or SN services within a given AA is exclusive. Therefore, it can be considered that there are two, different server xDLMS_ASE-s: one providing services with Logical name references and another providing services with Short name references. The server application layer shall include one or both of these xDLMS_ASEs.

NOTE A server could use both LN and SN referencing in different AAs.

On the client side, in order to handle the different referencing schemes transparently for the COSEM client AP, the COSEM client application layer provides only one service set, using Logical Name referencing. This has two major consequences:

- using a unique, standardized service set between COSEM client applications and the communications protocol – hiding the particularities of different COSEM servers – 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 (e.g. Windows XP, UNIX, etc.) Using this – public – API specification, client applications can be developed without knowledge about particularities of a given server;

- when the COSEM server device does not use LN referencing, the client application layer shall include an additional component. The purpose of this component is to map the LN service set, used by the client AP into/from the service set, used by the server AP. Figure 6 shows the COSEM client application layer when the server is using SN referencing. The additional component is called SN_MAPPER_ASE. See also 6.5.5.2.

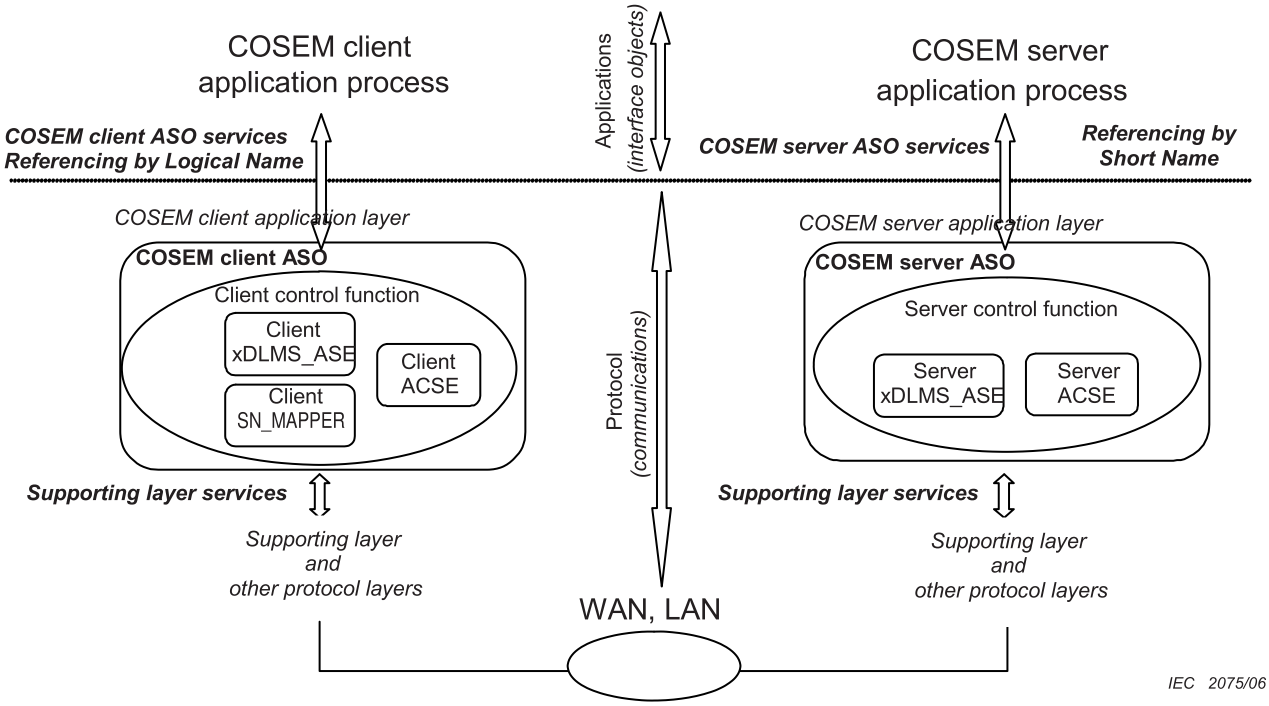

COSEM client application process
Applications (interface objects)
COSEM server application process
COSEM client ASO services Referencing by Logical Name
COSEM server ASO services
Referencing by Short Name
COSEM client application layer
COSEM client ASO
Client control function
Client xDLMS_ASE
Client SN_MAPPER
Client ACSE
Protocol (communications)
COSEM server application layer
COSEM server ASO
Server control function
Server xDLMS_ASE
Server ACSE
Supporting layer services
Supporting layer and other protocol layers
Supporting layer services
Supporting layer and other protocol layers
WAN, LAN
IEC 2075/06

Figure 6 – Structure of the COSEM AL when the server is using SN references

5.4 Layer management services

Layer management services have local importance only. Therefore, specification of these services is not within the scope of this standard. The specific SetMapperTables service is defined in 6.5.5.1.

5.5 Protocol specification

The COSEM application layer protocols specify the procedures for the transfer of information for application association control, authentication (ACSE procedures) and for data exchange between COSEM clients and servers (xDLMS procedures). These procedures are defined in terms of:

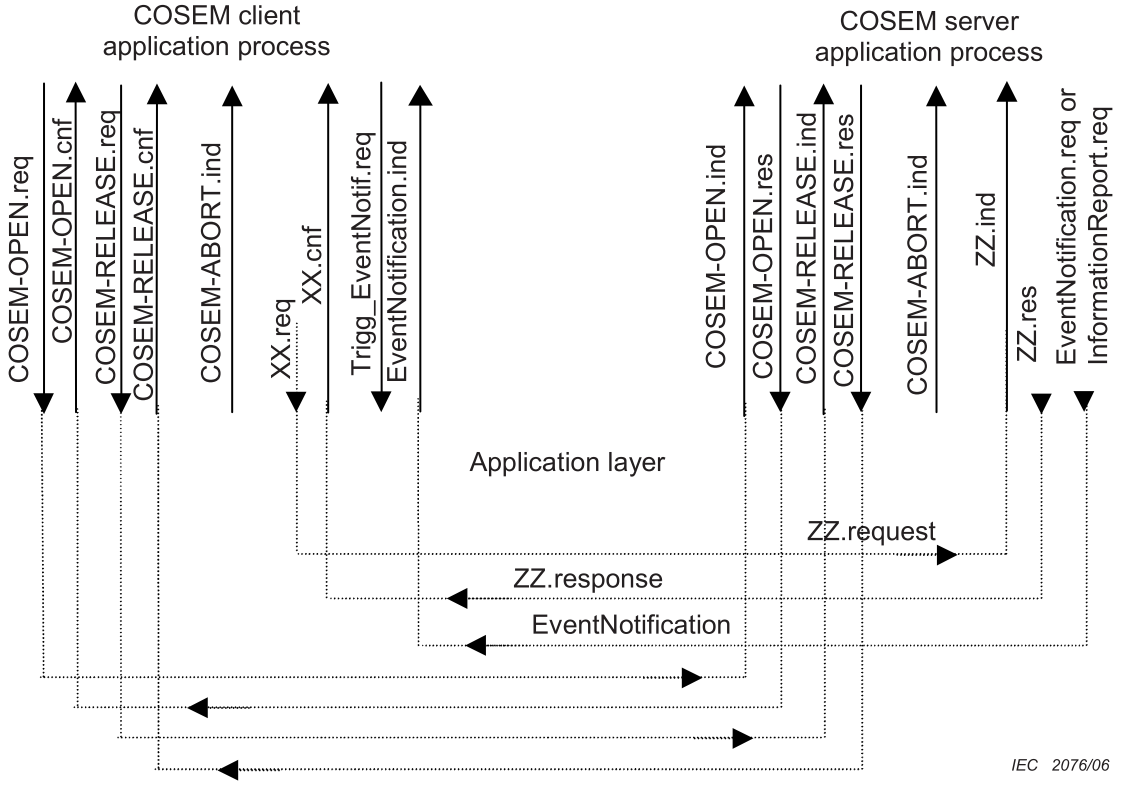
- the interactions between peer ACSE and xDLMS protocol machines through the use of services of the supporting protocol layer;
- the interactions between the ACSE and xDLMS protocol machines and their service user;
- the abstract syntax (ASN.1, ISO/IEC 8824) representation of Application Protocol Data Units (APDUs) is also specified with the application protocols; see Clause 8.

NOTE All COSEM services are operating on an already established physical connection. Establishment of this physical connection is done outside of the COSEM protocol, therefore, it is not within the scope of this standard.

6 COSEM application layer – Service specification

6.1 Summary of services

A summary of the services available at the top of the COSEM application layer is shown in Figure 7.



NOTE XX and ZZ refer to client/server type data communication services. These services may be different on the client side and the server side, if the server does not use LN referencing. See 6.4.

Figure 7 – Summary of COSEM application layer services

6.2 Application association establishment and release

The COSEM-OPEN, COSEM-RELEASE and COSEM-ABORT services are used for the establishment and release of AAs.

The COSEM-OPEN.request service is invoked by the COSEM client AP to establish a confirmed or non-confirmed AA with a COSEM server AP. Invoking this service implies generating a COSEM-OPEN.indication service primitive at the server side[6]. If the association to be established is a confirmed one, the server shall respond to this request by invoking the COSEM-OPEN.response service, which is transferred to the client AP as a remote confirmation (COSEM-OPEN.confirm). This normal opening sequence is shown in Figure 8.

[6] Before the invocation of the COSEM-OPEN.request service, the physical layers must be connected. Depending on the communication profile, the invocation of the COSEM-OPEN.request service may also imply the connection of the lower layers.

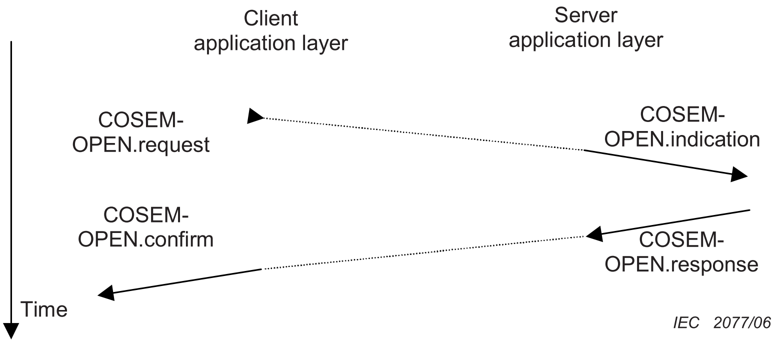


Figure 8 – Normal service sequence for the COSEM-OPEN service

NOTE The COSEM-OPEN.request service may also be locally (negatively) confirmed, for example when the connection of a lower layer is not successful.

The COSEM-RELEASE service is provided for graceful disconnection of an existing application association. As COSEM server application processes are not allowed to request a graceful disconnection, the COSEM-RELEASE.request service is available only for the COSEM client. The nominal service sequence for the COSEM-RELEASE service is the same as shown in Figure 8 for the COSEM-OPEN service, replacing the word ‘OPEN’ with the word ‘RELEASE’.

The ABORT service is used to indicate the disconnection of the physical connection. This service is the same at both sides.

6.3 Special application associations

6.3.1 Confirmed application associations

For the purposes of this standard, the term *confirmed application association* is used to designate an AA, which is established between a client and a server AP with the help of an AARQ/AARE message exchange (see 7.3.1). Establishment of a confirmed AA is always initiated by the client application in invoking the COSEM-OPEN.request service with Service_Class == Confirmed.

After the establishment of a confirmed AA, any xDLMS data communication services using LN referencing can be invoked in a confirmed or an unconfirmed manner, until the association is released.

NOTE xDLMS services using SN referencing are either confirmed (Read, Write) or Unconfirmed (Unconfirmed Write).

6.3.2 Non-confirmed application associations

For the purposes of this standard, the term *non-confirmed application association* is used to designate an AA, which is established without an AARQ/AARE message exchange: only an AARQ shall be sent from the client to the server. Similarly to the confirmed AA, establishment of a non-confirmed AA is also always initiated by the Client application, but in invoking the COSEM-OPEN.request service with Service_Class == Unconfirmed.

After the establishment of a non-confirmed AA, xDLMS data communication services using LN referencing can only be invoked in unconfirmed manner, until the association is released.

NOTE With SN referencing, in non-confirmed AAs only the UnconfirmedWrite service can be used.

As the establishment of such non-confirmed AAs does not require the Server AP to respond to the association request coming from the client, in some cases – for example one-way communications or broadcasting – the establishment of a non-confirmed AA is the only possibility.

6.3.3 Pre-established application associations

Pre-established AAs do not need to be established using the COSEM-OPEN service. It can be considered, that this COSEM-OPEN has already been done (it does not matter how). Consequently, pre-established associations can be considered existing from the moment the lower layers are able to deliver APDUs between the client and the server.

A pre-established AA can be either confirmed or non-confirmed (depending on the way it is pre-established), but in any case it cannot be released. The purpose of this type of association is to simplify data exchange with simple devices (e.g. supporting one-way communication only). The pre-established AA eliminates the need of connection establishment and release (phases 1 and 3 in Figure 4) and only data communication services are used. These must use connectionless services of the supporting lower protocol layers[7].

6.3.4 Mandatory application associations

The mandatory management logical device in the physical metering device must support an AA with a public client, with the lowest security level.

In any communication profile, the management logical device and the public client must have a reserved identifier/address.

6.4 Data communication

For data communication purposes, the client application layer provides the following set of services (referred to as XX in Figure 7).

- GET service (.request,.confirm);
- SET service (.request,.confirm);
- ACTION service (.request,.confirm).

All these services refer to attributes or methods of COSEM interface objects via logical names.

Received erroneous confirmed service requests are normally simply discarded at the server side. However, in that case, COSEM servers may optionally respond with an EXCEPTION-Response APDU (see 8.6.1), indicating that the previously received service request cannot be correctly processed.

There are also non-client/server type services to support receiving information like alarms from a COSEM server without first requesting it by the client. These are:

- EventNotification service (.indicate);
- Trigger_EventNotification_Sending (.request).

In confirmed AAs, the client application layer obtains knowledge about the referencing method used by the server during the AA establishment phase. In case of a pre-established AA, the client application layer is expected to know the referencing method used by the server before data communication services can be used. When the client AP invokes data communication services, the application layer shall invoke the services and send the APDUs corresponding to the referencing style used by the server (referred to as ZZ in Figure 7).

[7] Pre-established application associations are not possible in profiles, where the supporting lower protocol layer(s) do not provide connectionless data communication services. As for all application associations, the logical devices must contain an Association LN /SN interface object for the pre-established associations, too.

When the server is also using LN references, the server side service set is the complementary of the client side service set (the same service set, but.request services shall be transferred as.indication services, and the.confirm services are originated as.response services).

When the server is using SN references, the service set is as follows:

- READ service (.indication,.response);
- WRITE service (.indication,.response);
- UNCONFIRMED WRITE service (.indication);
- InformationReport service (.request).

As explained in 5.3.2, in order to able to 'map' between the different service sets, the client application layer shall include an additional protocol component, called 'Client SN_MAPPER'.

The corresponding server application layer shall signal the reception of this (LN or SN referencing) APDU to the server AP. In most cases, the server AP responds to the received.request service by invoking the corresponding.response service. Upon the reception of the APDU, corresponding to that.response invocation, the client application layer shall generate the appropriate logical name referencing service primitive to the client AP.

6.5 Client COSEM application layer services

6.5.1 Application association establishment

6.5.1.1 Overview

Figure 9 shows services provided by the client side application layer for AA establishment. These services are provided by the ACSE.

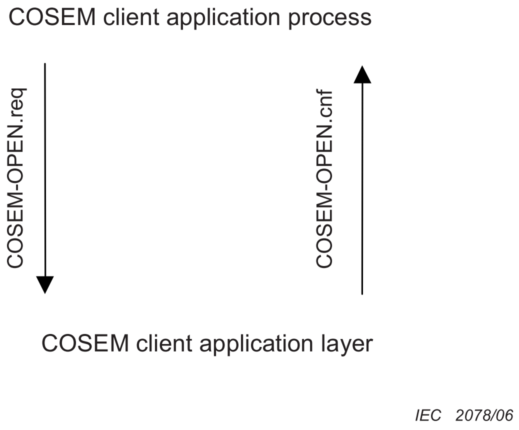

COSEM client application process
COSEM-OPEN.req
COSEM-OPEN.cnf
COSEM client application layer
IEC 2078/06

Figure 9 – Client side services for application association establishment

6.5.1.2 COSEM-OPEN.request

Function

This service is invoked by the COSEM client AP to request the establishment of an AA to a remote COSEM server AP.

Service parameters

The semantics of the primitive is as follows:

```
COSEM-OPEN.request
(
        Protocol_Connection_Parameters,
        Dedicated_Key,
        DLMS_Version_Number,
        DLMS_Conformance,
        Client_Max_Receive_PDU_Size,
        ACSE_Protocol_Version,
        Application_Context_Name,
        Application_Ids_and_Titles,
        Security_Mechanism_Name,
        Calling_Authentication_Value,
        Implementation_Information,
        User_Information,
        Service_Class
)
```

The Protocol_Connection_Parameters parameter contains all information necessary to use a lower layer profile, including the communication profile (protocol) identifier and the required addresses. Examples for this parameter are given in Annex B.

The Dedicated_Key, DLMS_Version_Number, DLMS_Conformance and Client_Max_ Receive_PDU_Size parameters contain respectively the value of the dedicated-key, the proposed-dlms-version-number, the proposed-conformance and the client-max-receive-pdu-size parameters of the xDLMS-Initiate.request PDU. These parameters are specified in IEC 61334-4-41, and in 8.4 of this standard. Annex C gives some examples of their usage. The xDLMS-Initiate.request PDU shall be inserted in the user-information field of the AARQ APDU to be sent.

The ACSE_Protocol_Version, Application_Context_Name, Application_Ids_and_Titles, Security_ Mechanism_Name and the Calling_Authentication_Value parameters shall be inserted into the corresponding fields of the AARQ APDU to be sent.

The xDLMS-ASE and the ACSE provide only the framework for *transporting* this information. To provide and verify that information is the job of the appropriate COSEM AP. Default and allowed values for these fields are defined in 7.3.3.

The Implementation_Information parameter is optional. If present, it shall be inserted into the implementation-information field of the AARQ APDU to be sent.

The User_Information parameter is optional. If present, it shall be passed on to the supporting layer.

The Service_Class parameter indicates whether the service shall be invoked in the confirmed or unconfirmed manner.

Use

The client AP uses this service to initiate the establishment of an AA to a remote server AP.

If the Client AP invokes a COSEM-OPEN.request service with a parameter referring to an already established AA, then the application layer shall locally and negatively confirm this request with the reason that the requested AA already exists. Note, that this is always the case for pre-established AAs.

When the *Protocol_Connection_Parameters* parameter indicates a connection-oriented communication profile (e.g. TCP/IP) but the Service_class parameter is set to *Unconfirmed*, the COSEM application layer shall locally and negatively confirm this request, with the reason that the requested AA is not allowed.

When the *Protocol_Connection_Parameters* parameter indicates that one or more supporting lower layer needs to be connected and the requested AA is allowed, the COSEM client shall first establish all required lower layer connections (except for the physical layer connection, which must be already established prior to this service invocation).

When the required supporting lower layer services are available, the COSEM application layer shall construct and send an AARQ APDU to its peer, containing the service parameters received from the AP.

If the COSEM-OPEN.request service is invoked with Service_class == Confirmed, the *response-allowed* parameter of the xDLMS-InitiateRequest PDU, inserted in the user-information field of the constructed AARQ shall be set to TRUE. The client application layer is waiting for an AARE response from the server, prior to – positively or negatively – confirming the COSEM-OPEN.request service invocation.

If the COSEM-OPEN.request service is invoked with Service_Class == Unconfirmed, the *response-allowed* parameter shall be set to FALSE, the client application layer does not wait any response from the server. In this case, the service invocation shall be locally confirmed.

The protocol for AA establishment is specified in 7.3.1.

6.5.1.3 COSEM-OPEN.confirm

Function

This service is invoked by the COSEM client application layer to indicate whether the previously requested AA is accepted or not.

Service parameters

The semantics of the primitive is as follows:

```
COSEM-OPEN.confirm
(
        Protocol_Connection_Parameters,
        Local_or_Remote,
        Result,
        Failure_type,
        DLMS_Version_Number,
        DLMS_Conformance,
        Server_Max_Receive_PDU_Size,
        ACSE_Protocol_Version,
        Application_Context_Name,
        Application_Ids_and_Titles,
        Security_Mechanism_Name,
        Responding_Authentication_Value,
        Implementation_Information
)
```

The Protocol_Connection_Parameters parameter contains all the information required to identify the protocol connection having been established. These parameters identify the participants of the AA requested by the preceding COSEM-OPEN.request service.

The Local_or_Remote parameter indicates the origin of the COSEM-OPEN.confirm service primitive invocation. When this parameter is set to Remote, the service invocation has been originated by the reception of an AARE APDU from the remote server. Otherwise, the service is locally originated.

In case of a remote confirmation, the Result parameter indicates whether the COSEM server AP accepted the requested association or not. In case of local confirmation, the Result parameter indicates whether the client side protocol stack accepted the request or not. In the case of non-acceptance (remote or local), the Failure_type parameter indicates the reason for not accepting the proposed association.

The DLMS_Version_Number, DLMS_Conformance and Server_Max_Receive_PDU_Size parameters contain respectively the value of the negotiated-dlms-version-number, negotiated-conformance and server-max-receive-PDU-size parameters of the xDLMS-Initiate.response PDU. These parameters are specified in IEC 61334-4-41, and in 8.4 of this standard. Annex C gives some examples for their usage. The xDLMS-Initiate.response PDU is transported in the user-information field of the received AARE APDU.

The ACSE_Protocol_Version, Application_Context_Name, Application-Ids_and_Titles, Security_ Mechanism_Name and the Responding_Authentication_Value parameters carry the value of the corresponding fields of the received AARE APDU.

The Implementation_Information parameter, if present, carries the value of the implemen–tation-information field of the received AARE APDU.

Use

The COSEM client application layer uses this service primitive to indicate to the client AP whether the previously proposed AA is accepted or not. It may be generated as a result of a received AARE APDU (remote confirmation). It may also be generated locally in the following cases:

- when the requested AA already exists (this case includes pre-established AAs);
- when the corresponding COSEM-OPEN.request has been invoked with Service_class == Unconfirmed;
- when the requested AA is not allowed;
- due to a locally detected error (missing or not correct parameters, failure during the establishment of the requested lower layer connections, missing physical connection, etc.

6.5.2 Application association release

6.5.2.1 Overview

Figure 10 shows the services provided by the client application layer for releasing an existing AA.

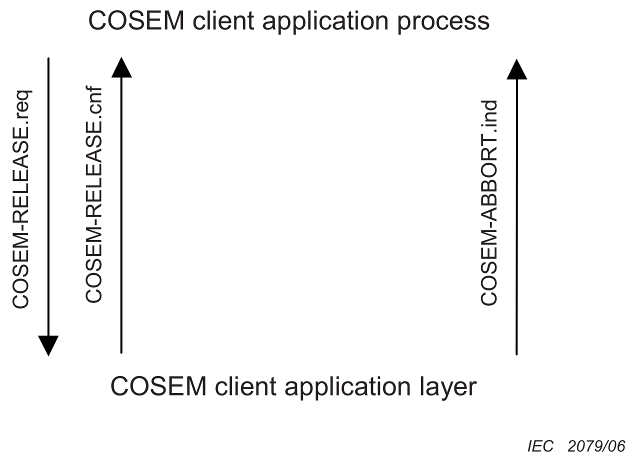

COSEM client application process
COSEM-RELEASE.req
COSEM-RELEASE.cnf
COSEM-ABBORT.ind
COSEM client application layer
IEC 2079/06

Figure 10 – Client side services for releasing an application association

In certain COSEM communication profiles – for example in the 3-layer, connection-oriented, HDLC based profile – there is a one-to-one relationship between a confirmed AA and the supporting protocol layer connection. In this case, the COSEM-RELEASE services used during the association release phase do not rely on the ACSE A_RELEASE services. Confirmed AAs in these profiles are released simply by disconnecting the corresponding lower layer connection.

This is the mandatory way to release such AAs: in order to request that, the Client application shall invoke the COSEM-RELEASE.request service with no Use_RLRQ_RE parameter or with Use_RLRQ_RE == FALSE.

Note, that as this request shall imply the disconnection of the lower layer connection, there is no mandatory APDU associated to the COSEM-RELEASE service.

The client AP is informed about the result of the requested disconnection via the COSEM-RELEASE.confirm service primitive.

Optionally, the COSEM client AP may invoke the COSEM-RELEASE.request service with Use_RLRQ_RE == TRUE. In this case, the client application layer, instead of disconnecting the lower layer connection (if there is one), shall initiate the release of the referenced AA by sending an RLRQ APDU to the server which may respond to that with an RLRE APDU. The client AP is informed about the result of the association release via the COSEM-RELEASE.confirm service primitive, which, in this case, could be a remotely confirmed service.

Supporting the RLRQ/RLRE APDUs is optional in HDLC based profile.

Any existing AA – except the pre-established ones on the server side – shall be aborted, when the physical connection is intentionally disconnected or if any of the supporting layer connection is disconnected or breaks. A local COSEM-ABORT.indication primitive is provided to inform the AP about this. As physical connection/ disconnection is done outside of the protocol, requesting a physical disconnection is not within the scope of this standard.

6.5.2.2 COSEM-RELEASE.request

Function

This service primitive is invoked by the COSEM client AP to request the release of an existing AA with a remote COSEM server AP.

Service parameters

The semantics of this service primitive is as follows:

```
COSEM-RELEASE.request
(
        Use_RLRQ_RE,
        User_Information
)
```

The Use_RLRQ_RE parameter is optional. If it is not present, it shall be considered as if its value would be equal to FALSE.

The User_Information parameter is optional. If present, it shall be passed on to the supporting layer. Specification of the content of this parameter is not within the scope of this standard.

Use

The Client AP uses this service primitive to gracefully release an existing AA.

When this service is invoked with no Use_RLRQ_RE parameter or with Use_RLRQ_RE == FALSE, the invocation of this service shall not imply sending an APDU.

In communication profiles where the RLRQ service is not supported or only optionally supported, upon the reception of this service invocation the client application layer shall:

- either initiate the disconnection of the corresponding lower layer connection (if the AA to be released is a confirmed AA) by invoking the corresponding XX-DISCONNECT.request service of the supporting lower protocol layer;
- or shall simply release the requested non-confirmed AA and locally confirm the request.

Supporting the COSEM-RELEASE.request service with no Use_RLRQ_RE or with Use_RLRQ_RE == FALSE is mandatory for all COSEM Clients. However, in communication profiles where the RLRQ service is mandatory, invoking the COSEM-RELEASE.request service with no Use_RLRQ_RE or with Use_RLRQ_RE == FALSE may lead to an error[8]: in that case it shall be locally and negatively confirmed.

When this service primitive is invoked with Use_RLRQ_RE == TRUE, the client application layer shall construct an A-RELEASE.request (RLRQ) APDU, and shall send it to the peer server application layer, using the supporting lower layer services. If the RLRQ is sent within a non-confirmed AA, after sending the RLRQ the client application layer shall locally and positively confirm the current request. Otherwise, after sending the RLRQ APDU, the client application layer shall start a time-out, waiting for either the A-RELEASE.response (RLRE) APDU from the server, or for the time-out to be elapsed. On any of these two events, the client application layer shall confirm the COSEM-RELEASE.request primitive.

Supporting the COSEM-RELEASE.request service with Use_RLRQ_RE == TRUE is optional in HDLC based profile.

The protocol for releasing an AA is described in 7.3.6.

6.5.2.3 COSEM-RELEASE.confirm

Function

The COSEM client application layer invokes this service primitive to indicate to the AP whether the previously received request for releasing the AA is accepted.

[8] When the lower layer connections are not managed by the COSEM application layer.

NOTE The server cannot refuse a release request.

Service parameters

The semantics of the primitive is as follows:

```
COSEM-RELEASE.confirm
(
        Result,
        Failure_type,
        User_Information
)
```

The Result parameter is the report of the corresponding COSEM-RELEASE.request service. As servers cannot refuse requests of neither releasing AAs nor disconnecting lower layer connections, the value of the Result parameter should normally be SUCCESS.

However, the value of this parameter may also be ERROR: in that case, the Failure_type parameter indicates the reason for that.

The User_Information field may be present only when the service is remotely confirmed. In this case, it contains user specific information carried by the supporting lower protocol layer(s), if this is possible. Specification of its content is not within the scope of this standard.

Use

The COSEM client application layer uses this service primitive to indicate to the client AP the result of the previously requested release of an AA. This service primitive is originated:

- as a result of the invocation of a XX-DISCONNECT.confirm service (where XX is the supporting lower protocol layer); or
- by a locally detected error – missing or not correct parameters, or communication failure at lower protocol layer level; or
- after sending out an RLRQ APDU within a non-confirmed AA (OPTIONAL); or
- by the time-out on waiting for an RLRE APDU from the remote Server (OPTIONAL); or
- as a result of a RLRE APDU received from the remote Server (OPTIONAL).

6.5.2.4 COSEM-ABORT.indication

Function

This service is invoked by the client application layer to indicate to the client AP an unsolicited disconnection of any supporting lower layer connection, including the disconnection of the physical layer.

Service parameters

The semantics of the primitive is as follows:

```
COSEM-ABORT.indication
(
        Diagnostics
)
```

The optional Diagnostics parameter 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 standard.

Use

The client application layer uses this service primitive to indicate to the COSEM client AP that a lower layer connection abort occurred in a non-solicited manner (e.g. the physical line is broken).

NOTE In communication profiles, where the supporting layer connection is not managed by the COSEM application layer, a disconnection of the supporting layer by the manager process leads also to a COSEM-ABORT.indication.

6.5.3 Client/server type data communication services

6.5.3.1 Service overview

Figure 11 shows services provided by the client side application layer during the data communications phase.

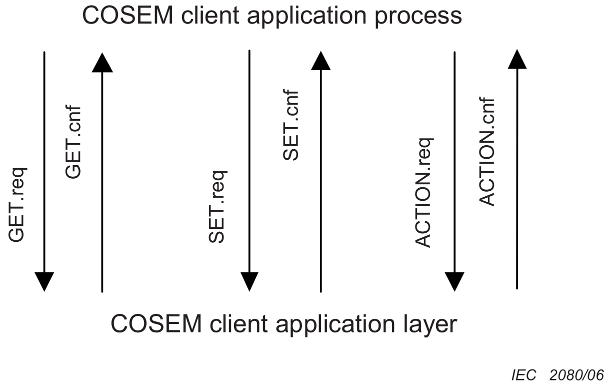

COSEM client application process
GET.req
GET.cnf
SET.req
SET.cnf
ACTION.req
ACTION.cnf
COSEM client application layer
IEC 2080/06

Figure 11 – Client side data communication services

Data communication services rely on the services of the xDLMS_ASE. These services contain references to attributes or methods of COSEM interface objects.

For COSEM servers, two types of referencing are specified in IEC 62056-62: Logical Name (LN) and Short Name (SN). The COSEM client application layer provides only one service set, using logical name referencing. Consequently, when the COSEM server device does not use logical name referencing, the client application layer shall include an additional application protocol component; see in Figure 6. The purpose of this is to ‘map’ the LN service set into/from the service set used by the server AP.

The service set provided at the COSEM client side is:

- COSEM interface object attribute related services: GET, SET (.request,.confirm);
- COSEM interface object method related service: ACTION (.request,.confirm).

The.request primitive of these services is invoked by the COSEM client AP. The role of the protocol with regard to these services is to transport them as.indication to the COSEM server AP.

NOTE Consequently, a.request APDU is identical to an.indication APDU and a.response APDU is identical to a.confirm APDU. For APDU definitions, see 8.6.

All data communication services within a confirmed AA can be invoked in a confirmed or non-confirmed manner. In case of non-confirmed AAs, data communication services may only be invoked in a non-confirmed manner.

In case of confirmed service invocation, the server AP shall return the confirmation by invoking the corresponding.response service primitive. The receipt of this response is indicated to the client AP via the.confirm service primitive.

If a confirmed service request cannot be processed by the server (e.g. the request has been received without establishing an AA first, or the request is otherwise erroneous) normally it is discarded by the application layer. However, the COSEM server application layer may optionally send an EXCEPTION-Response APDU to indicate to the client application layer that the service request received could not be processed and the reasons for this.

The optional EXCEPTION-Response APDU is defined in 8.6.1.

Unconfirmed service invocation will not imply.response/.confirm primitive invocation. The reason for this is to avoid collisions due to potential multiple responses in the case of multicasting and/or broadcasting.

The protocol for confirmed service invocations is described in 7.4.1.1 and for unconfirmed service invocations in 7.4.1.2.

6.5.3.2 GET.request

Function

This service is invoked by client AP to request the value(s) of one or all attributes of one or more COSEM interface object(s) from the remote server AP.

Service parameters

The semantics of the primitive is as follows:

```
GET.request
(
        Invoke_Id,
        Priority,
        Service_Class,
        Request_Type,
        COSEM_Attribute_Descriptor, { COSEM_Attribute_Descriptor,},
        Block_Number
)
COSEM_Attribute_Descriptor
(
        COSEM_Class_Id,
        COSEM_Object_Instance_Id,
        COSEM_Object_Attribute_Id,
        Access_Selection_Parameters
)
```

The Invoke_Id parameter identifies the instance of this service invocation.

The value of the Priority parameter indicates the priority level associated to the received request. There are two priority levels: normal (FALSE) and high (TRUE).

The Service_Class parameter indicates whether the service is invoked in confirmed or unconfirmed manner.

The Request_type parameter indicates the type of the current GET.request service invocation: NORMAL, NEXT or WITH-LIST. A GET.request always starts with a GET.request type NORMAL or WITH-LIST. A GET.request with NEXT type is issued only when the requested data is too long for being transferred in one.response APDU. The protocol for non-transparent long data transfer with the GET service is described in 7.4.1.8.2.

A GET.request service shall contain one or more COSEM_Attribute_Descriptor parameters, each of them referencing one or all attributes of a COSEM interface object. The COSEM_Attribute_Descriptor parameter is a composite parameter, consisting of the following components:

- the { COSEM_Class_Id, COSEM_Object_Instance_Id } doublet non-ambiguously identifies one and only one COSEM interface object instance;
- the COSEM_Object_Attribute_Id component identifies the attribute(s) of the object instance. COSEM_Object_Attribute_Id = 0 references all attributes of the designated object instance;
- the optional Access_Selection_Parameters component, in case of selective access (see 7.4.1.6) carries the additional data required for the selective GET operation. This parameter can be used only when COSEM_Object_Attribute_Id != 0.

One GET.request invocation may contain as many COSEM_Attribute_Descriptors as the server-max-receive-pdu-size allows. The COSEM_Attribute_Descriptor(s) shall be present only with Request_type == NORMAL or WITH-LIST.

The optional Block_Number parameter is present only when Request_type == NEXT. It carries the number of the last correctly received block of long data.

Use

The client AP uses this service primitive to request the value(s) of one or all attributes of one or more COSEM interface object(s) from the server AP.

6.5.3.3 GET.confirm

Function

This service is invoked by the client application layer to indicate the reception of a Get-response-XX APDU from the COSEM server AP.

Service parameters

The semantics of the primitive is as follows:

```
GET.confirm
(
    Invoke_Id,
    Priority,
    Response_type,
    Result, { Result, }
    Block_Number
)
```

The Invoke_Id parameter identifies the instance of this service invocation. Its value shall be equal to the Invoke_Id of the corresponding GET.request service invocation.

The value of the Priority parameter indicates the priority level associated to the response received. The value of this parameter shall be equal to the value of the Priority parameter of the corresponding GET.request service invocation.

The Response_type parameter indicates whether this.confirm service invocation contains the complete response to the previous GET.request service invocation, or it contains only a block of the required data. This parameter shall carry one of the following values:

- NORMAL: the service invocation contains the complete response for a NORMAL GET.request;

- WITH-LIST: the service invocation contains the complete response for a GET.request service of type WITH-LIST (including a list of attribute references);
- ONE-BLOCK: the service invocation contains one block of the complete response. The Block_Number parameter carries the number of the data block carrying a part of the result as raw data;
- LAST-BLOCK: the service invocation contains the last data block of the response.

The Result parameter shall carry either the requested data, or in case of error, the indication of the type of error. If the encoded form of the Result parameter does not fit in one APDU, then it shall be transported in blocks, carried by the result parameter of the Get-Confirm-With-Datablock APDU, of type DataBlock-G. This parameter shall include the block number and the encoded form of the result as raw data or data access result.

The number of Result parameters in the GET.confirm service shall be the same as the number of COSEM_Attribute_Descriptor parameters in the corresponding GET.request service – one response for each request.

Use

The client application layer uses this service primitive to indicate the reception of a Get-Response-XX APDU.

6.5.3.4 SET.request

Function

This service primitive is invoked by the client AP to request the remote server AP to set the value of one or more attributes of a COSEM interface object.

Service parameters

The semantics of the primitive is as follows:

```
SET.request
(
        Invoke_Id,
        Priority,
        Service_Class,
        Request_type,
        COSEM_Attribute_Descriptor, { COSEM_Attribute_Descriptor, },
        Block_Number,
        Data, { Data, }
)

COSEM_Attribute_Descriptor
(
        COSEM_Class_Id,
        COSEM_Object_Instance_Id,
        COSEM_Object_Attribute_Id,
        Access_Selection_Parameters
)
```

The Invoke_Id parameter identifies the instance of this service invocation.

The value of the Priority parameter indicates the priority level associated to the received request. There are two priority levels: normal (FALSE) and high (TRUE).

The Service_Class parameter indicates whether the service is invoked in confirmed or unconfirmed manner.

The Request_type parameter indicates whether the Data parameter of the service primitive carries all the data necessary to set all the attributes referenced by the COSEM_ Attribute_Descriptor (list) or only a block of it. It shall be set to one of the following values:

- NORMAL: the service invocation contains the reference to one or all (Attribute_0 feature, see 7.4.1.7.1) attribute(s) of one COSEM interface object and all the required data. The optional Block_Number parameter shall not be present in the service invocation;
- WITH-LIST: the service invocation contains a list of COSEM interface object attribute references and all the required data. The optional Block_Number parameter shall not be present in the service invocation;
- FIRST-BLOCK: the service invocation contains the reference to one or all attribute(s) of one COSEM interface object and the first part of the required data. The Block_Number parameter shall be set to 0001;
- FIRST-BLOCK-WITH-LIST: the service invocation contains a list of COSEM interface object attribute references and the first part of the required data. The Block_Number parameter shall be set to 0001;
- ONE-BLOCK: the service invocation contains only one block of the data. The Block_Number parameter carries the number of the datablock carrying a part of the Data parameter as raw data, and no COSEM_Attribute_Descriptor(s) shall be present;
- LAST-BLOCK: the service invocation contains the last block of the Data. The Block_Number parameter carries the number of this data block, and no COSEM_ Attribute_Descriptor(s) shall be present.

NOTE In the case of ONE-BLOCK and LAST-BLOCK Set-Request-With-Datablock APDU is generated.

A SET.request service shall contain one or more COSEM_Attribute_Descriptor parameters, each of them referencing one or more attributes of a COSEM interface object. The COSEM_Attribute_Descriptor parameter is a composite parameter, consisting of the following components:

- the { COSEM_Class_Id, COSEM_Object_Instance_Id } doublet non-ambiguously identifies one and only one COSEM interface object instance;
- the COSEM_Object_Attribute_Id component identifies the attribute(s) of the object instance. COSEM_Object_Attribute_Id = 0 references all attributes of the designated object instance.

The optional Access_Selection_Parameters element, in case of selective access (see 7.4.1.6) carries the additional data required for the selective SET operation. This parameter can be used only when COSEM_Object_Attribute_Id != 0.

One SET.request invocation may contain as many COSEM_Attribute_Descriptors as the server-max-receive-pdu-size allows. The COSEM_Attribute_Descriptor(s) shall be present only when Request_type == NORMAL, Request_type == WITH-LIST or Request_type == FIRST-BLOCK-XXX.

The optional Block_Number parameter is present when Request_type != NORMAL or WITH-LIST. It carries the number of the data block within the current service invocation.

The Data parameter contains the data necessary to set the attributes identified by the Attribute_descriptor parameter(s). If the encoded form of the data does not fit in one APDU, then it shall be transported in blocks, carried by the datablock parameter of the appropriate Set-Request-XX APDU, of type DataBlock-SA. This parameter shall include the block number and the encoded form of the data as raw data. The protocol for long data transfer with the SET service is described in 7.4.1.8.3.

The number of Data parameters in the SET.request service shall be the same as the number of COSEM_Attribute_Descriptors: one Data for each COSEM_Attribute_Descriptor.

Use

The client AP uses this service primitive in order to request the remote server AP to set the value of one or more attributes of one or more COSEM interface objects.

6.5.3.5 SET.confirm

Function

This service primitive is invoked by the client application layer to indicate the reception of a SET.response from the COSEM server AP.

Service parameters

The semantics of the primitive is as follows:

```
SET.confirm
(
        Invoke_Id,
        Priority,
        Response_type,
        Result { Result, },
        Block_Number
)
```

The Invoke_Id parameter identifies the instance of this service invocation. Its value is equal to the Invoke_Id of the corresponding SET.request service invocation.

The value of the Priority parameter indicates the priority level associated to the received response. The value of this parameter is equal to the value of the Priority parameter of the corresponding SET.request service invocation.

The Response_type parameter indicates whether this.confirm service invocation contains the response for the complete SET.request operation, or it is simply an acknowledge of the previously received data block. This parameter shall carry one of the following values:

- NORMAL: the.confirm service contains the confirmation of the previous SET.request operation, which carried a single COSEM interface object attribute reference. The Result parameter carries the result of the required operation;
- WITH-LIST: the.confirm service contains the confirmation of the previous SET.request operation, which carried a list of COSEM interface object attribute references. The Result parameter carries the list of results for each required SET operation;
- ACK-BLOCK: this value indicates that this.confirm service contains the acknowledgement for the last correctly received data block. The Block_Number parameter carries the number of the received data block;
- LAST-BLOCK: the SET.confirm service is invoked with this value after the reception of the last data block of a SET.request service, which carried a reference to a single COSEM interface object attribute. This value indicates that this.confirm service contains the response to the original SET.request service, which has been sent in several blocks. The Result parameter carries the result of the required operation and the Block_Number parameter carries the number of the last data block;
- LAST-BLOCK-WITH-LIST: the SET.confirm service is invoked with this value after the reception of the last data block of a SET.request service, which carried a list of COSEM interface object attribute references. This value indicates that this.confirm contains the response to the original SET.request service, which has been sent in several blocks. The Result parameter carries the list of result for each required set operation and the Block_Number parameter carries the number of the last data block.

The number of Result parameters in the SET.confirm service with Response_type == WITH-LIST and LAST-BLOCK-WITH-LIST shall be the same as the number of attribute references in the corresponding SET.request service – one result for each request. Each Result parameter shall carry the result of the corresponding SET.request operation.

Use

The client application layer uses this service primitive to indicate the reception of a Set-Response-XX APDU.

6.5.3.6 ACTION.request

Function

This service is invoked by the client AP to remotely invoke one or more method(s) of one or more COSEM interface object(s) in the remote server AP.

Service parameters

The semantics of the primitive is as follows:

```
ACTION.request
(
        Invoke_Id,
        Priority,
        Service_Class,
        Request_Type,
        COSEM_Method_Descriptor, { COSEM_Method_Descriptor, },
        Block_Number,
        Method_Invocation_Parameters, { Method_Invocation_Parameters, }
)

COSEM_Method_Descriptor
(
        COSEM_Class_Id,
        COSEM_Object_Instance_Id,
        COSEM_Object_Method_Id
)
Method_Invocation_Parameters::= Data
```

The Invoke_Id parameter identifies the instance of this service invocation.

The value of the Priority parameter indicates the priority level associated to the received request. There are two priority levels: normal (FALSE) and high (TRUE).

The Service_Class parameter indicates whether the service is invoked in a confirmed or an unconfirmed manner.

The Request_type parameter indicates whether the given invocation contains a complete request or only a part of it. It shall be set to one of the following values:

- NORMAL: the service invocation contains the reference to one COSEM interface object method and the Method_Invocation_Parameters required for the invocation of this method. The optional Block_Number parameter shall not be present in the service invocation;
- WITH-LIST: the service invocation contains a list of COSEM interface object(s) method references and all the required Method_Invocation_Parameters. The optional Block_Number parameter shall not be present in the service invocation;

- FIRST-BLOCK: the service invocation contains the reference to one COSEM interface object method and the first part of the required Method_Invocation_Parameters. The Block_Number parameter shall be set to 0001;
- WITH-LIST-AND-FIRST-BLOCK: the service invocation contains a list of COSEM interface object methods and the first part of the required Method_Invocation_ Parameters. The Block_Number parameter shall be set to 0001;
- ONE-BLOCK: the service invocation contains only one block of the Method_Invocation_Parameters. The Block_Number parameter carries the number of the parameter block carrying a part of the Method_Invocation_Parameters parameter, and no COSEM_ Method_Descriptor shall be present;
- LAST-BLOCK: this value indicates that the current block is the last parameter block to be transferred. The Block_Number parameter carries the number of this parameter block, and no COSEM_Method_Descriptor(s) shall be present;
- NEXT: this value indicates that this.request contains an acknowledgement for a previously received parameter block, and requests the server to send the next one. The Block_Number parameter carries the number of the last correctly received parameter block.

An ACTION.request service shall contain one or more COSEM_Method_Descriptor parameters, each of them referencing one method of a COSEM interface object. The COSEM_Method_Descriptor parameter is a composite parameter, consisting of the following components:

The {COSEM_Class_Id, COSEM_Object_Instance_Id} doublet non-ambiguously identifies one and only one COSEM interface object instance. The complete COSEM_ Method_Descriptor references one method of that object instance: this method is identified by the COSEM_Object_Method_Id component.

The optional Block_Number parameter is present either when the.request contains a parameter block to be sent or when the.request acknowledges a previously received parameter block (Request_type == NEXT). The Block_Number parameter carries the number of the last received parameter block.

Invoking a method may require additional parameters. The Method_Invocation_Parameters parameter carries the data necessary for the invocation of the method(s) identified by the COSEM_Method_Descriptor parameter. If the encoded form of the Method_ Invocation_Parameters does not fit in one APDU, then it shall be transported in blocks, carried by the pblock parameter of the appropriate Action-Request-XX APDU, of type DataBlock-SA. This parameter shall include the block number and the encoded form of the Method_Invocation_Parameters as raw data.

The ACTION.request service shall contain as many Method_Invocation_Parameters then COSEM_Method_Descriptors: one Method_Invocation_Parameters for each COSEM_ Method_Descriptor. Therefore, even if the invocation of a method does not require additional parameters, the corresponding Method_Invocation_Parameters component shall be present in the service invocation – but it shall be empty.

The COSEM_Method_Descriptor parameter shall not be present when Request_type == ONE-BLOCK or LAST BLOCK.

Use

This service primitive is used by the client AP to remotely invoke one or more method(s) of one or more COSEM interface object(s) in the remote server AP.

6.5.3.7 ACTION.confirm

Function

This service is invoked by the client application layer to indicate the reception of a ACTION.response from the COSEM server AP.

Service parameters

The semantics of the primitive is as follows:

```
ACTION.confirm
(
        Invoke_Id,
        Priority,
        Response_type,
        Result, { Result, },
        Block_Number,
        Response_Parameters, { Response_Parameters, }
)
```

The Invoke_Id parameter identifies the instance of this service invocation. Its value shall be equal to the Invoke_Id of the corresponding ACTION.request service invocation.

The value of the Priority parameter indicates the priority level associated to the received response. The value of this parameter shall be equal to the value of the Priority parameter of the corresponding ACTION.request service invocation.

The Response_type parameter indicates whether this.confirm service invocation contains the complete response requested by the previous ACTION.request service invocation, it contains only a block of the required data, or it is simply an acknowledge of a previously received block of the ACTION.request service. This parameter shall carry one of the following values:

- NORMAL: the service invocation contains the complete response for a NORMAL ACTION.request which carried a single COSEM interface object method reference;
- WITH-LIST: the service invocation contains the complete response for a WITH-LIST ACTION.request service, including a list of COSEM interface object method references;
- ONE-BLOCK: the service invocation contains only one block of the complete response. The Block_Number parameter carries the number of the parameter block carrying a part of the response as raw data;
- LAST-BLOCK: this value indicates that the service invocation contains the last block of the response as raw data;
- NEXT: this value indicates that service invocation contains an acknowledgement for the previously received parameter block and requests the client to send the next one. The Block_Number parameter carries the number of the last correctly received parameter block.

The Result parameter carries the result of the invocation of the COSEM interface object method(s).

The Response_Parameters carries the optional data to be returned, as a result of the invocation of the COSEM interface object methods.

The number of Result and Response_Parameters parameters in the ACTION.confirm service primitive with Response_type == WITH-LIST or a.confirm service which is sent in several parameter blocks shall be the same as the number of COSEM interface object method references in the corresponding ACTION.request service – one Result and Response_ Parameter for each request.

If the encoded form of the Result and Response_Parameters does not fit in one APDU, then it shall be transported in block, carried by the pblock parameter of the Action-Response-With-Pblock APDU, of type DataBlock-SA. This parameter shall include the block number and the encoded form of the Result and Response_parameters as raw data.

Use

The client application layer uses this service primitive to indicate the reception of an Action-Response-XX APDU.

6.5.4 Client side services for event notification

Figure 12 shows services provided by the client side application layer for event notification.

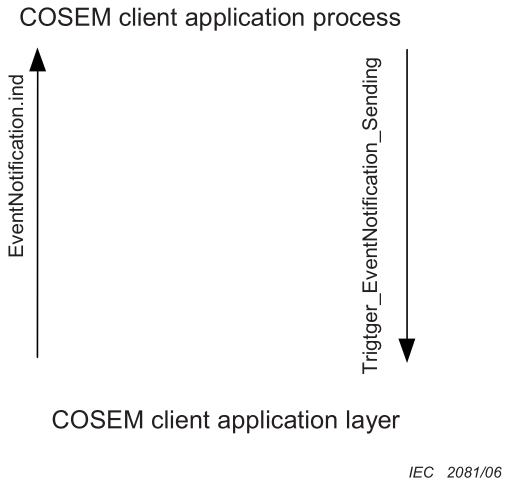

COSEM client application process
EventNotification.ind
Trigtger_EventNotification_Sending
COSEM client application layer
IEC 2081/06

Figure 12 – Client side services for event notification

The EventNotification service is the only non-client/server type service provided in COSEM. Using the EventNotification.request service, the server AP is able to send an unsolicited notification of the occurrence of an event to the remote client AP. Reception of the EventNotification message is indicated to the client AP via the EventNotification.indication primitive. The protocol is described in 7.4.1.3.

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 shall explicitly solicit sending an EventNotification frame, by invoking the Trigger_EventNotification_sending service primitive.

6.5.4.1 EventNotification.indication

Function

This service is invoked by the client application layer to indicate the reception of an EVENT-NOTIFICATION.indication from the COSEM server SAP.

Service parameters

The semantics of the primitive is as follows:

```
EventNotification.indication
(
        Time,
        Application-Addresses,
        COSEM_Attribute_Descriptor,
        Attribute_Value
)
```

```
COSEM_Attribute_Descriptor
(
        COSEM_Class_Id,
        COSEM_Object_Instance_Id,
        COSEM_Object_Attribute_Id
)
```

The optional Time parameter indicates the time assigned to the event by the server.

The Application_Addresses parameter is optional. It is present only when the EventNotification A-PDU is received outside of an established AA. In this case, it contains all protocol specific parameters required to identify the sender and destination APs.

The { COSEM_Class_Id, COSEM_Object_Instance_Id, COSEM_Object_Attribute_Id } triplet identifies 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

The client application layer uses this service primitive to indicate the reception of an EVENT-NOTIFICATION.indication to the AP.

6.5.4.2 Trigger_EventNotification_Sending.request

Function

This service is invoked by the client in order to trigger the server to send the frame carrying the EVENT-NOTIFICATION-Request APDU.

NOTE This service is necessary in case of communication profiles, when the server is not able to send a real non-solicited EventNotification message.

Service parameters

The semantics of the primitive is as follows:

```
Trigger_EventNotification_Sending.req
(
        Protocol_Parameters
)
```

The Protocol_Parameters parameter contains all lower protocol dependent information, which is required for triggering the server to send out an eventually pending frame containing an EVENT-NOTIFICATION-Request APDU. This information includes the protocol identifier, and all the required lower layer parameters.

Use

Upon the reception of a Trigger_EventNotification_Sending.request service invocation from the client AP, the client application layer shall invoke the corresponding supporting layer service to send a trigger message to the server.

6.5.5 Client side layer management services

This subclause defines a special layer management service, used to manage the short name mapper application service element. This client side service is necessary only if the server uses SN referencing. All other layer management services are not within the scope of this standard.

6.5.5.1 SetMapperTable.request

Function

This 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. This service is necessary only if on the server side SN referencing is used.

Service parameters

The semantics of the primitive is as follows:

```
SetMapperTable.request
(
        Mapping_table
)
```

The Mapping_table parameter contains the contents of the attribute "object_list" for the requested server and AA. The structure of the content is defined in IEC 62056-62.

Use

The client AP uses this service primitive, in order to enhance the efficiency of the mapping process if SN referencing is used.

6.5.5.2 Mapping client services for servers using Short names

For servers using SN referencing, the services listed above are mapped to the corresponding xDLMS services (compare IEC 61334-4-41) by the client Control function (see Figure 6) in the following manner:

Table 1 – Mapping between client side LN and server side SN referencing services

Client side xDLMS Service (LN ref.)	Server side xDLMS Service (SN ref.)
GET.request	ReadRequest
GET.confirm	ReadResponse
SET.request (Service_Class="confirmed")	WriteRequest
SET.request (Service_Class="unconfirmed")	UnconfirmedWriteRequest
SET.confirm	WriteResponse
ACTION.request (Service_Class="unconfirmed")	UnconfirmedWriteRequest
ACTION.request (Service_Class="confirmed")	Action with return parameters: ReadRequest VariableAccessSpecification:= parametrised access (IEC 61334-4-41) Selector:= 0; If no method invocation parameters are supplied: Parameter:= null-data Action without return parameters: WriteRequest If no method invocation parameters are supplied: Data:= null-data
ACTION.confirm	ReadResponse If no data is returned then: data:= null-data.
EVENTNOTIFICATION.indication	InformationReportRequest

Details about the mapping of the logical names to short names are given in IEC 62056-62.

6.6 Server COSEM application layer services

6.6.1 Application association establishment

6.6.1.1 Overview

Figure 13 shows the services provided by the server application layer for AA establishment. These services are provided by the ACSE.

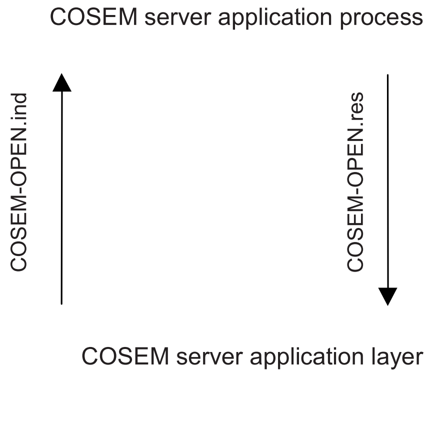

COSEM server application process
COSEM-OPEN.ind
COSEM-OPEN.res
COSEM server application layer
IEC 2082/06

Figure 13 – Server side services for application association establishment

6.6.1.2 COSEM-OPEN.indication

Function

This service is invoked by the server side of the application layer following the receipt of an AARQ APDU, to indicate to the COSEM server AP that the peer (client) AP requested the establishment of an AA.

Service parameters

The semantics of the primitive is as follows:

```
COSEM-OPEN.indication
(
        Protocol_Connection_Parameters,
        Dedicated _Key,
        DLMS_Version_Number,
        DLMS_Conformance,
        Client_Max_Receive_PDU_Size,
        ACSE_Protocol_Version,
        Application_Context_Name,
        Application-Ids_and_Titles,
        Security_Mechanism_Name,
        Calling_Authentication_Value,
        Implementation_Information,
        User_Information,
        Service_Class
)
```

The Protocol_Connection_Parameters parameter contains all information necessary to use the supporting layer, including the profile (protocol) identifier and the required addresses. Examples for this parameter are given in Annex B.

The Dedicated_Key, DLMS_Version_Number, DLMS_Conformance and Client_Max_Receive _PDU_Size parameters contain respectively the value of the dedicated-key, the proposed-dlms-version-number, the proposed-conformance and the client-max-receive-pdu-size parameters of the xDLMS-Initiate.request PDU. These parameters are specified in IEC 61334-4-41 and in 8.4 of this standard. Annex C gives some examples for their usage. The xDLMS-Initiate.request PDU shall be inserted in the user-information field of the AARQ APDU received.

The ACSE_Protocol_Version, Application_Context_Name, Application_Ids_and_Titles, Security_Mechanism_Name and the Calling_Authentication_Value parameters are carried by the corresponding fields of the received AARQ APDU.

The xDLMS-ASE and the ACSE provide only the framework for transporting this information. To provide and verify that information is the job of the appropriate COSEM AP. Default and allowed values for these fields are defined in 7.3.7.

The Implementation_Information parameter, if present, carries the value of the implementation-information field of the received AARQ APDU.

The User_Information parameter is optional. When present, it contains the information sent by the Client AP using the same parameter in the corresponding.request primitive.

The Service_Class parameter indicates whether the service is invoked in confirmed or unconfirmed manner.

Use

This service is used by the server side application layer to indicate the reception of a correctly formatted AARQ APDU to the COSEM server AP. In order to be able to receive this APDU, lower layer protocol connections – if it is required – have to be already established.

The protocol for AA establishment is described in 7.3.1.

6.6.1.3 COSEM-OPEN.response

Function

This service is invoked by the server AP to indicate whether the previously proposed AA is accepted or not.

Service parameters

The semantics of the primitive is as follows:

```
COSEM-OPEN.response
(
      Protocol_Connection_Parameters,
      Result,
      Failure_type,
      DLMS_Version_Number,
      DLMS_Conformance,
      Server_Max_Receive_PDU_Size,
      ACSE_Protocol_Version,
      Application_Context_Name,
      Application_Ids_and_Titles,
      Security_Mechanism_Name,
      Responding_Authentication_Value,
      Implementation_Information
 )
```

The Protocol_Connection_Parameters parameter contains all information necessary to use the supporting layer, including the profile (protocol) identifier and the required addresses. Examples for this parameter are given in Annex B.

The Result parameter indicates whether the COSEM server AP accepted the association request or not.

In the case of non-acceptance, the Failure_type parameter indicates the reason for not accepting the proposed AA.

The DLMS_Version_Number, DLMS_Conformance and Server_Max_Receive_PDU_Size parameters contain respectively the value of the negotiated-dlms-version-number, negotiated-conformance and server-max-receive-pdu-size parameters of the xDLMS Initiate.response PDU. These parameters are specified in IEC 61334-4-41, and in 8.4 of this standard. Annex C gives some examples for their usage. The xDLMS-Initiate.request PDU shall be inserted in the user-information field of the AARE APDU to be sent.

The ACSE_Protocol_Version, Application_Context_Name, Application-Ids_and_Titles, Security_ Mechanism_Name and the Responding_Authentication_Value parameters shall be inserted into the corresponding fields of the AARE APDU to be sent.

The Implementation_Information parameter, if present, shall be inserted in the implementation-information field of the AARE APDU to be sent.

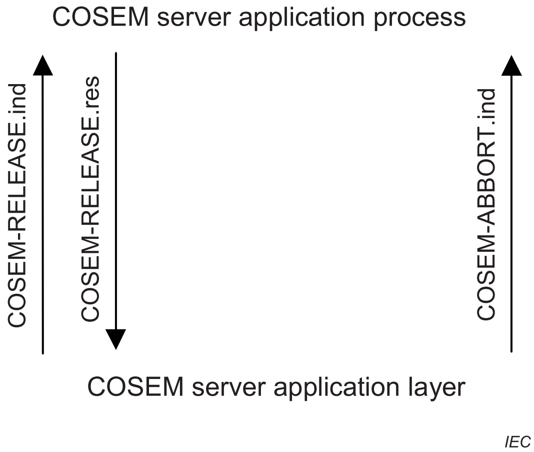
Use

This service primitive is used by the COSEM server AP to indicate to the application layer whether the previously proposed AA is accepted or not.

If the designated AA is confirmed (it was requested by an AARQ with response-allowed == TRUE), the invocation of this service shall imply sending an AARE APDU to the requestor client application layer, otherwise no AARE shall be sent.

6.6.2 Application association release

Figure 14 shows the services provided by the server side application layer for disconnecting an AA.


COSEM server application process
COSEM-RELEASE.ind
COSEM-RELEASE.res
COSEM-ABBORT.ind
COSEM server application layer

IEC 2083/06

Figure 14 – Server side services for releasing an application association

In certain COSEM communication profiles – for example in the 3-layer, connection-oriented, HDLC based profile – there is a one-to-one relationship between a confirmed AA and the supporting protocol layer connection. In this case, the COSEM-RELEASE and COSEM-ABORT services used during the association release phase do not rely on the ACSE services. Confirmed AAs in these profiles are released simply by disconnecting the corresponding lower layer connection.

This is the mandatory way to release such AAs.

This way of releasing an AA is requested by the client AP by invoking the COSEM-RELEASE.request service with no Use_RLRQ_RE parameter or with Use_RLRQ_RE == FALSE. In receiving this request, the client application layer shall disconnect the supporting layer connection.

Disconnecting the supporting layer connection shall imply the server application layer to receive an XX-DISCONNECT.indication with REASON == REMOTE from the supporting protocol layer. The reception of this service makes the server application layer to indicate the request for releasing the corresponding AA to the server AP with the help of the COSEM-RELEASE.indication service primitive. The server application shall respond to this with the invocation of the COSEM-RELEASE.response primitive. This response will imply the server application layer to respond to the XX-DISCONNECT.indication and release the designated AA.

Optionally, the COSEM client AP may invoke the COSEM-RELEASE.request service with Use_RLRQ_RE == TRUE. In this case, the client application layer, instead of disconnecting the supporting layer connection (if there is one), shall initiate the release of the referenced AA by sending an RLRQ APDU to the Server.

The reception of this RLRQ APDU has the same effect as the XX-DISCONNECT.indication: the server application layer shall indicate the received release request to the server AP with the help of the COSEM-RELEASE.indication service primitive. The server AP shall respond to this with the COSEM-RELEASE.response.

If the AA to be released is a confirmed AA, the reception of this COSEM-RELEASE.response will imply sending an RLRE APDU to the Client, otherwise no RLRE shall be sent.

Supporting the RLRQ/RLRE APDUs is optional.

Any existing AA – except the pre-established ones on the server side – shall be aborted, when the physical connection is intentionally disconnected or if any of the supporting layer connection is disconnected or breaks. A local COSEM-ABORT.indication primitive is provided to inform the AP about this. As physical connection/disconnection is done outside of the protocol, requesting a physical disconnection is not within the scope of this standard.

6.6.2.1 COSEM-RELEASE.indication

Function

This service primitive is invoked by the COSEM server application layer to indicate to the server AP a supporting layer disconnection indication.

Service parameters

The semantics of the primitive is as follows:

```
COSEM-RELEASE.indication
(
        Use_RLRQ_RE,
        User_Information
)
```

The Use_RLRQ_RE parameter is optional. If it is not present, it is considered, as if the value of this parameter would be equal to FALSE.

The User_Information parameter is optional. When it is present, it shall contain User-specific information carried by the supporting lower protocol layer(s). Specification of the contents of this parameter is not within the scope of this standard.

Use

This service is used by the server application layer to indicate to the server AP that a graceful release of the AA has been requested. The server must accept this request.

When this service is invoked with no Use_RLRQ_RE parameter or with Use_RLRQ_RE == FALSE, the invocation of this service is generated upon the reception of a supporting protocol layer disconnect indication. In this case, the server AP shall respond to this indication with a COSEM-RELEASE.response primitive with no Use_RLRQ_RE or with Use_RLRQ_RE == FALSE, too.

Supporting the COSEM-RELEASE.indication service with no Use_RLRQ_RE or with Use_RLRQ_RE == FALSE is mandatory for all COSEM Servers.

Optionally, COSEM servers may support releasing AAs with the help of the RLRQ/RLRE APDUs. COSEM servers supporting this operation shall invoke the COSEM-RELEASE.indication service with Use_RLRQ_RE == TRUE on the receipt of an RLRQ APDU. In this case, if the COSEM-RELEASE.indication has been received in a confirmed AA, the server application shall respond to this indication with a COSEM-RELEASE.response primitive with Use_RLRQ_RE == TRUE, too.

6.6.2.2 COSEM-RELEASE.response

Function

This service primitive is invoked by the COSEM server AP to indicate to the application layer whether the previously received request for releasing the AA has been accepted.

NOTE The server cannot refuse a received request for disconnection.

Service parameters

The semantics of the primitive is as follows:

```
COSEM-RELEASE.response
(
        Result,
        User_Information,
        Use_RLRQ_RE
)
```

The Result parameter indicates whether the server AP can accept the previous COSEM-RELEASE.request or not. Its value depends on whether the AA, the release of which has been requested, was existing or not.

If the User_Information parameter is present, it shall be passed on to the supporting protocol layer. Specification of its content is not within the scope of this standard.

The Use_RLRQ_RE parameter is optional. If it is not present, it is considered, as if the value of this parameter would be equal to FALSE. If this parameter is present and its value is TRUE, then the primitive is a response to a COSEM-RELEASE.indication, which has been originated by the reception of a RLRQ APDU. Otherwise, the origin of the COSEM-RELEASE.indication was the reception of a disconnect indication with REASON == REMOTE from the supporting protocol layer.

Use

This service primitive is invoked by the server AP to respond to a previously received COSEM-RELEASE.indication. Upon the reception of this invocation, the server application layer shall process the release of the corresponding AA, and perform the appropriate actions as well. These actions can be one of the following:

- invoking the XX-DISCONNECT.response service of the supporting protocol layer, if the Use_RLRQ_RE parameter is not present or its value is FALSE;
- constructing and sending an RLRE APDU to the peer client application layer, if the Use_RLRQ_RE parameter is present and its value is TRUE, and the released AA is a confirmed one. Supporting this operation is optional.

6.6.2.3 COSEM-ABORT.indication

Function

This service primitive is invoked by the COSEM server application layer to indicate to the server application that the AA is aborted because of the supporting lower protocol connection is shut down. It can be the result either of an unsolicited disconnection of any supporting lower layer connection, including the disconnection of the physical layer, or of the action of another application entity, such as the connection manager application, present in some profiles.

Service parameters

The semantics of the primitive is as follows:

```
COSEM-ABORT.indication
(
        Diagnostics
)
```

The optional Diagnostics parameter shall indicate the possible reason for the physical disconnection, and can carry lower protocol layer dependent information. Specification of the contents of this parameter is not within the scope of this standard.

Use

The server application layer uses this service primitive to indicate to the COSEM server AP that a lower layer connection abort occurred in a non-solicited manner. The origin of the abort can be an external event (e.g. the physical line is broken), or an action of a supporting layer connection manager AP, present in some profiles.

6.6.3 Client/server type data communication services

6.6.3.1 Service overview

Services provided during the data communications phase rely on services of the xDLMS_ASE. These services contain references to attributes or methods of COSEM interface objects. IEC 62056-62 defines two different types of referencing, by logical name (LN) and by short name (SN). Therefore, two different server xDLMS_ASE-s – thus two different server application layers – are specified. These server side application layers provide two different sets of services. One set of services (GET, SET, ACTION and EventNotification) is using exclusively LN references. The other set of services (Read, Write, Unconfirmed Write, InformationReport) is using exclusively SN references.

However, during the lifetime of an established AA, there is only one server xDLMS_ASE present in the COSEM server application layer. The type of this xDLMS_ASE is negotiated during the connection establishment phase and only the selected xDLMS_ASE is present within the server application layer. It explains, why using one or the other set of services is exclusive. No Read/Write/UnconfirmedWrite services are provided by the COSEM server ASO when the AA is established within a context using LN referencing, and no GET/SET/ACTION/EventNotification services are provided in the opposite case.

6.6.3.2 Services provided with LN references

Figure 15 shows services provided by the server side application layer during the data communications phase, when LN referencing is used:

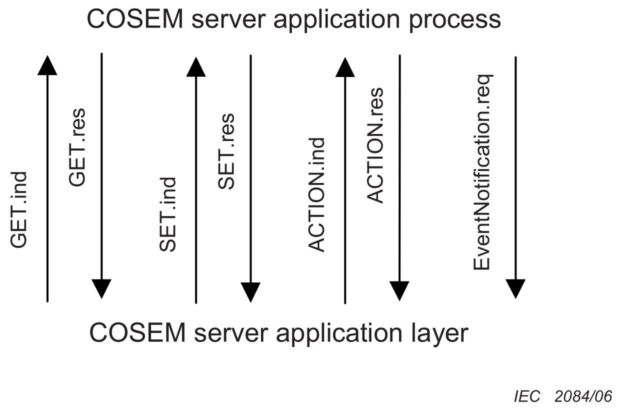


Figure 15 – Server side data communications services using LN referencing

Three client/server type services may be supported when LN referencing is used: GET, SET and ACTION. The.request primitive of these services is invoked by the COSEM client AP. The role of the protocol with regard to these services is to transport them to the COSEM server AP. The server application layer shall indicate the reception of a request via the.indication service primitive to the server AP.

Each of these services can be requested in a confirmed or an unconfirmed manner. However, in case of a non-confirmed AA, data communication services may only be invoked in a non-confirmed manner.

In case of confirmed service invocation, the server AP shall return the confirmation by invoking the corresponding.response service primitive. The receipt of this response is indicated to the client AP via the.confirm service primitive.

If a confirmed service request cannot be processed by the server (e.g. the request has been received without establishing an AA first, or the request is otherwise erroneous) normally, it is discarded by the application layer. However, the COSEM server application layer may optionally send an EXCEPTION-Response APDU to indicate to the client application layer that the service request received could not be processed and the reasons for this.

The optional EXCEPTION-Response APDU is defined in 8.6.1.

Unconfirmed service invocation will not imply.response/.confirm primitive invocation. In COSEM, the only reason to do it is to avoid collisions due to potential multiple responses in the case of multicasting and/or broadcasting.

The protocol for confirmed service invocations is described in 7.4.1.1 and for unconfirmed service invocations in 7.4.1.2.

The fourth, EventNotification Service is the only non-client/server service provided in COSEM. By invoking this service, the server AP is able to send an unsolicited notification of the occurrence of an event to the remote client.

6.6.3.2.1 GET.indication

Function

This service is invoked by the server application layer to indicate to the server AP that a remote client has requested the value(s) of one or all attributes of one or more COSEM interface object(s).

Service parameters

The semantics of the primitive is as follows:

```
GET.indication
(
        Invoke_Id,
        Priority,
        Service_Class,
        Request_type,
        COSEM_Attribute_Descriptor, { COSEM_Attribute_Descriptor, },
        Block_Number
)

COSEM_Attribute_Descriptor
(
        COSEM_Class_Id,
        COSEM_Object_Instance_Id,
        COSEM_Object_Attribute_Id,
        Access_Selection_Parameters
)
```

The Invoke_Id parameter identifies the instance of this service invocation.

The value of the Priority parameter indicates the priority level associated to the received request. There two priority levels: normal (FALSE) and high (TRUE).

The Service_Class parameter indicates whether the service is invoked in a confirmed or an unconfirmed manner.

The Request_type parameter indicates the origin and the type of the current GET.indication service invocation. It can be: NORMAL, WITH-LIST or NEXT.

The first GET.indication is always type NORMAL or WITH-LIST. It indicates the reception of a NORMAL GET.request from the client. A GET.indication with NEXT type indicates that the remote client is asking for the next data block. Non-transparent long data transfer with the GET service is defined in 7.4.1.8.2.

A GET.indication service shall contain one or more COSEM_Attribute_Descriptor parameters, each of them referencing a COSEM interface object attribute. The COSEM_Attribute_ Descriptor parameter is a composite parameter, consisting of the following components:

- the { COSEM_Class_Id, COSEM_Object_Instance_Id } doublet non-ambiguously identifies one and only one COSEM interface object instance;
- the COSEM_Object_Attribute_Id component identifies the attribute(s) of the object instance. COSEM_Object_Attribute_Id = 0 references all attributes of the designated object instance;
- the optional Access_Selection_Parameters element, in case of selective access (see 7.4.1.6.) carries the additional data required for the selective GET operation. This parameter can be used only when COSEM_Object_Attribute_Id != 0.

One GET.indication invocation may contain as many COSEM_Attribute_Descriptors as the server-max-receive-pdu-size allows. The COSEM_Attribute_Descriptor(s) shall be present only with Request_type == NORMAL or WITH-LIST.

The optional Block_Number parameter is present only when Request_type == NEXT. It carries the number of the last correctly received block of a long data, and no COSEM_Attribute_ Descriptor parameter shall be present.

Use

The server application layer generates the GET.indication service primitive upon the reception of a GET.request from the supporting layer.

6.6.3.2.2 GET.response

Function

This service is invoked by the server AP in order to send a response to a previously received GET.indication primitive.

Service parameters

The semantics of the primitive is as follows:

```
GET.response
(
        Invoke_Id,
        Priority,
        Response_type,
        Result, { Result, }
        Block_Number
)
```

The Invoke_Id parameter identifies the instance of this service invocation. Its value shall be equal to the Invoke_Id of the corresponding GET.indication service invocation.

The value of the Priority parameter indicates the priority level associated to the received.indication. The value of this parameter shall be equal to the value of the Priority parameter of the corresponding GET.indication service invocation.

The Response_type parameter indicates whether this.response service invocation contains the complete response requested by the previous GET.request service invocation, or it contains only a block of the required data. This parameter shall carry one of the following values:

- NORMAL: the service invocation contains the complete response for a NORMAL GET.request service;
- WITH-LIST: the service invocation contains the complete response for a WITH-LIST GET.request service;
- ONE-BLOCK: the service invocation contains only one block of the complete response. The Block_Number parameter carries the number of the data block carrying a part of the result as raw data;
- LAST-BLOCK: this value indicates that the current block is the last data block sent.

The Result parameter shall carry either the requested data, or in case of error, the indication of the type of error. If the encoded form of the Result parameter does not fit in one APDU, then it shall be transported in blocks, carried by the Result parameter of the appropriate Get-Response-With-Datablock APDU, of type DataBlock-G. This parameter shall include the block number and a part of the encoded form of the result as raw data or data access result.

The number of Result parameters in the GET.response service shall be the same as the number of COSEM_Attribute_Descriptor parameters in the corresponding GET.indication service – one response for each request.

Use

This service is used by the server AP. Upon the reception of the GET.response service invocation, the COSEM server application layer shall build a Get-Response-XX APDU. In case of success – when the corresponding GET.indication has been accepted – this APDU shall be built by encoding the received Data parameter otherwise the APDU will contain the value of the data_access_result parameter. In both cases, the Invoke_Id and the Priority parameter shall also be inserted into the APDU.

6.6.3.2.3 SET.indication

Function

This service primitive is invoked by the server application layer to indicate to the server AP that a remote client has requested setting one or more attributes of a COSEM interface object.

Service parameters

The semantics of the primitive is as follows:

```
SET.indication
(
        Invoke_Id,
        Priority,
        Service_Class,
        Request_type,
        COSEM_Attribute_Descriptor, { COSEM_Attribute_Descriptor, },
        Block_Number,
        Data, { Data, }
)
COSEM_Attribute_Descriptor
(
        COSEM_Class_Id,
        COSEM_Object_Instance_Id,
        COSEM_Object_Attribute_Id,
        Access_Selection_Parameters
)
```

The Invoke_Id parameter identifies the instance of this service invocation.

The value of the Priority parameter indicates the priority level associated to the received request. There are two priority levels: normal (FALSE) and high (TRUE).

The Service_Class parameter indicates whether the service is invoked in a confirmed or an unconfirmed manner.

The Request_type parameter indicates whether the Data parameter of the service primitive carries a complete attribute or only a block of it. It shall be set to one of the following values:

- NORMAL: the service invocation contains the reference to one or all (Attribute_0 feature, see 7.4.1.7.1) attribute(s) of one COSEM interface object and all the required data. The optional Block_Number parameter shall not be present in the service invocation;
- WITH-LIST: the service invocation contains a list of COSEM interface object attribute references and all the required data. The optional Block_Number parameter shall not be present in the service invocation;

- FIRST-BLOCK: the service invocation contains the reference to an attribute of one COSEM interface object and the first part of the required data. The Block_Number parameter shall be set to 0001;
- FIRST-BLOCK-WITH-LIST: the service invocation contains a list of COSEM interface object attribute references and the first part of the required data. The Block_Number parameter shall be set to 0001;
- ONE-BLOCK: the service invocation contains only one block of the data. The Block_Number parameter carries the number of the data block carried by the Data parameter, and no COSEM_Attribute_Descriptor(s) shall be present;
- LAST-BLOCK: this value indicates that the current is the last block of the data. The Block_Number parameter carries the number of this data block, and no COSEM_Attribute_Descriptor(s) shall be present.

A SET.indication service shall contain one or more COSEM_Attribute_Descriptor parameters, each of them referencing one or all attributes of a COSEM interface object. The COSEM_Attribute_Descriptor parameter is a composite parameter, consisting of the following components:

- the { COSEM_Class_Id, COSEM_Object_Instance_Id } doublet non-ambiguously identifies one and only one COSEM interface object instance;
- the COSEM_Object_Attribute_Id component identifies the attribute(s) of the object instance. COSEM_Object_Attribute_Id = 0 references all attributes of the designated object instance;
- the optional Access_Selection_Parameters element, in case of selective access (see 7.4.1.6.) carries the additional data required for the selective SET operation. This parameter can be used only when COSEM_Object_Attribute_Id != 0.

One SET.indication invocation may contain as many COSEM_Attribute_Descriptors as the server-max-receive-pdu-size allows. The COSEM_Attribute_Descriptor(s) shall be present only with Request_type == NORMAL, Request_type == WITH-LIST or Request_type == FIRST-BLOCK-XXX.

The optional Block_Number parameter is present when Request_type != NORMAL or WITH-LIST. It carries the number of the DataBlock within the current service invocation.

The Data parameter contains the data necessary to set the attributes identified by the Attribute_descriptor parameter. If the encoded form of the data does not fit in one APDU, then it shall be transported in blocks, carried by the datablock parameter of the appropriate Set-Indication-XX APDU, of type DataBlock-SA. This parameter shall include the block number and a part of the encoded form of the data as raw data.

The number of Data parameters in the SET.request service shall be the same as the number of COSEM_Attribute_Descriptors: one Data for each COSEM_Attribute_Descriptor.

Use

The server application layer generates the SET.indication service primitive upon the reception of a SET.request from the supporting layer.

6.6.3.2.4 SET.response

Function

This service primitive is invoked by the server AP to send a response to a previously received SET.indication primitive.

Service parameters

The semantics of the primitive is as follows:

```
SET.response
(
        Invoke_Id,
        Priority,
        Response_type,
        Result { Result, },
        Block_Number
)
```

The Invoke_Id parameter identifies the instance of this service invocation. Its value shall be equal to the Invoke_Id of the corresponding SET.indication service invocation.

The value of the Priority parameter indicates the priority level associated to the received response. The value of this parameter shall be equal to the value of the Priority parameter of the corresponding SET.indication service invocation.

The Response_type parameter indicates whether this.response service invocation contains the response for the complete SET.indication operation, or it is simply an acknowledge of the previously received data block. This parameter shall carry one of the following values:

- NORMAL: the.response service contains the confirmation of the previous SET.indication operation, which carried a single COSEM interface object attribute reference. The Result parameter carries the result of the required operation;
- WITH-LIST: the.response service contains the confirmation of the previous SET.request operation, which carried a list of COSEM interface object attribute references. The Result parameter carries the list of results for each required set operation;
- ACK-BLOCK: this value indicates that this.response contains a positive or negative acknowledgement for a previously received data block. The Block_Number parameter carries the number of the last correctly received data block;
- LAST-BLOCK: the SET.response service is invoked with this value after the reception of the last block of the data of a SET.request service, which carried a single COSEM interface object attribute reference. This value indicates that this.response contains the response to the original SET.indication service, which has been transferred in several blocks. The Result parameter carries the result of the required operation and the Block_Number parameter carries the number of the last data block;
- LAST-BLOCK-WITH-LIST: the SET.response service is invoked with this value after the reception of the last block of the data of a SET.request service, which carried a list of COSEM interface object attribute references. This value indicates that this.response contains the response to the original SET.indication service, which has been transferred in several blocks. The Result parameter carries the list of results for each required set operation and the Block_Number parameter carries the number of the last data block.

The number of the Result parameters in the SET.response service primitive with Response_type == WITH-LIST and LAST-BLOCK-WITH-LIST shall be the same as the number of COSEM interface object attribute references in the corresponding SET.request service – one result for each request. Each Result parameter shall carry the result of the corresponding SET.request operation.

Use

This service is used by the server AP. Upon the reception of the SET.response service invocation, the COSEM server application layer shall build a Set-Response-XX APDU. This APDU shall contain the response(s) for the corresponding SET.request – one Data-Access-Result parameter for each attribute set request. In case of success, this parameter shall contain a positive acknowledgement for the required set operation otherwise its value shall indicate the reason of the failure. In both cases, the Invoke_Id and the Priority parameters shall also be inserted into the APDU.

6.6.3.2.5 ACTION.indication

Function

This service is invoked by the server application layer to indicate to the server AP that a remote client has requested the invocation of one or more methods of one or more COSEM interface objects.

Service parameters

The semantics of the primitive is as follows:

```
ACTION.indication
(
        Invoke_Id,
        Priority,
        Service_Class,
        Request_Type,
        COSEM_Method_Descriptor, { COSEM_Method_Descriptor, },
        Block_Number,
        Method_Invocation_Parameters, { Method_Invocation_Parameters, }
)

COSEM_Method_Descriptor
(
        COSEM_Class_Id,
        COSEM_Object_Instance_Id,
        Method_Id
)

Method_Invocation_Parameters::= Data
```

The Invoke_Id parameter identifies the instance of this service invocation.

The value of the Priority parameter indicates the priority level associated to the received request. There are two priority levels: normal (FALSE) and high (TRUE).

The Service_Class parameter indicates whether the service is invoked in a confirmed or an unconfirmed manner.

The Request_type parameter indicates whether the given invocation contains a complete request or only a part of it. It shall be set to one of the following values:

- NORMAL: the service invocation contains the reference to a method of one COSEM interface object and the Method_Invocation_Parameters required for the invocation of this method. The optional Block_Number parameter shall not be present in the service invocation;
- WITH-LIST: the service invocation contains a list of COSEM interface object methods and all the required Method_Invocation_Parameters. The optional Block_Number parameter shall not be present in the service invocation;
- FIRST-BLOCK: the service invocation contains the reference to a method of one COSEM interface object and the first part of the required Method_Invocation_Parameters. The Block_Number parameter shall be set to 0001;
- WITH-LIST-AND-FIRST-BLOCK: the service invocation contains a list of COSEM interface object method references and the first part of the required Method_Invocation_Parameters. The Block_Number parameter shall be set to 0001;

- ONE-BLOCK: the service invocation contains only one block of the Method_ Invocation_Parameters. The Block_Number parameter carries the number of the parameter block carrying a part of the Method_Invocation_Parameters parameter, and no COSEM_Method_Descriptor(s) shall be present;
- LAST-BLOCK: this value indicates that the current block is the last parameter block to be transferred. The Block_Number parameter carries the number of the parameter block carrying the last block of the Method_Invocation_Parameters and no COSEM_ Method_Descriptor(s) shall be present;
- NEXT: this value indicates that this.request contains an acknowledgement for a previously received parameter block. The Block_Number parameter carries the number of the last correctly received parameter block.

An ACTION.indication service shall contain one or more COSEM_Method_Descriptor parameters, each of them referencing one COSEM interface object method. The COSEM_Method_Descriptor parameter is a composite parameter, consisting of the following components:

- the { COSEM_Class_Id, COSEM_Object_Instance_Id } doublet non-ambiguously identifies one and only one COSEM interface object instance. The complete COSEM_ Method_Descriptor references one method of that object instance: this method is identified by the COSEM_Object_Method_Id component;
- the optional Block_Number parameter is present either when the.indication contains a parameter block to be sent or when the.request acknowledges a previously received parameter block (Request_type == NEXT). The Block_Number parameter carries the number of the last received parameter block.

Invoking a method may require additional parameters. The Method_Invocation_Parameters parameter carries the data necessary for the invocation of the method(s) identified by the COSEM_Method_Descriptor parameter(s). If the encoded form of the Method_ Invocation_Parameters does not fit in one APDU, then it shall be transported in blocks, carried by the pblock parameter of the appropriate Action-Indication-XX APDU, of type DataBlock-SA. This parameter shall include the block number and a part of the encoded form of the Method_Invocation_Parameters as raw data.

The ACTION.indication service shall contain as many Method_Invocation_Parameters as COSEM_Method_Descriptors: one Method_Invocation_Parameter for each COSEM_Method_ Descriptor. Therefore, even if the invocation of a method does not require additional parameters, the corresponding Method_Invocation_Parameters component shall be present in the service invocation – but it shall be empty.

The COSEM_Method_Descriptors parameter shall not be present when Request_type == ONE-BLOCK or LAST BLOCK.

Use

The server application layer generates the ACTION.indication service primitive upon the reception of an Action-Request APDU from the supporting layer.

6.6.3.2.6 ACTION.response

Function

This service primitive is invoked by the server AP to send a response to a previously received ACTION.indication primitive.

Service parameters

The semantics of the primitive is as follows:

ACTION.response

```
(
        Invoke_Id,
        Priority,
        Response_type,
        Result, { Result, },
        Block_Number,
        Response_Parameters, { Response_Parameters, }
)
```

The Invoke_Id parameter identifies the instance of this service invocation. Its value shall be equal to the Invoke_Id of the corresponding ACTION.indication service invocation.

The value of the Priority parameter indicates the priority level associated to the received.response. The value of this parameter shall be equal to the value of the Priority parameter of the corresponding ACTION.indication service invocation.

The Response_type parameter indicates whether this.response service invocation contains the complete response requested by the previous ACTION.indication service invocation, it contains only a block of the required data, or it is simply an acknowledge of a previously received block of the ACTION.indication service. This parameter shall carry one of the following values:

- NORMAL: the service invocation contains the complete response for a NORMAL Action.indication which carried a single method reference;
- WITH-LIST: the service invocation contains the complete response for a WITH-LIST ACTION.indication service, including a list of COSEM interface object method references;
- ONE-BLOCK: the service invocation contains only one block of the complete response. The Block_Number parameter carries the number of the data block carrying a part of the response as raw data;
- LAST-BLOCK: this value indicates that this.response primitive contains the last block of the response as raw data;
- NEXT: this value indicates that this.response contains an acknowledgement for a previously received parameter block and requests the client to send the next one. The Block_Number parameter carries the number of the last correctly received parameter Block.

The Result parameter carries the result of the invocation of the COSEM interface object method(s).

The Response_Parameters carries the optional data to be returned as a result of the invocation of the COSEM interface object methods.

The number of Result and Response_Parameters parameters in the ACTION.confirm service primitive with Response_type == WITH-LIST or a.confirm service which is sent in several parameter blocks shall be the same as the number of COSEM interface object method references in the corresponding ACTION.request service – one Result and Response_ Parameter for each request.

If the encoded form of the Result and Response_Parameters does not fit into one APDU, then they shall be transported in blocks, carried by the pblock parameter of the Action-Response-With-Pblock APDU, of type DataBlock-SA. This parameter shall include the block number and the encoded form of the Result and Response_parameters as raw data.

Use

This service is used by the server AP. Upon the reception of the ACTION.response service invocation, the COSEM server application layer shall build an Action-Response-XX APDU.

This shall include the result and optionally, when the ACTION service has to return data, the return parameters.

The result parameter may be success, or eventually include the reason for the failure.

The return parameters may include the data to be returned, or if the data cannot be accessed, the reason for the access failure.

If the action response does not fit in one APDU, it shall be sent back in blocks of type DataBlock-SA, with the help of the transparent or non-transparent long data transfer mechanism. These mechanisms are defined in 7.4.1.8.

In all cases, the Invoke_Id and the Priority parameters shall also be inserted into the APDU.

6.6.3.2.7 EventNotification.request

Function

This service is invoked by the server AP to send an EventNotification message to the remote client AP.

Service parameters

The semantics of the primitive is as follows:

```
EventNotification.request
(
        Time,
        Application_Addresses,
        COSEM_Attribute_Descriptor,
        Attribute_Value
)
COSEM_Attribute_Descriptor
(
        COSEM_Class_Id,
        COSEM_Object_Instance_Id,
        COSEM_Object_Attribute_Id
)
```

The optional Time parameter indicates the time assigned to the event by the server.

The Application_Addresses parameter is optional. It is present only when the EventNotification A-PDU is to be sent outside of an established AA. In this case, it contains all protocol specific parameters required to identify the source and destination APs.

The { COSEM_Class_Id, COSEM_Object_Instance_Id, COSEM_Object_Attribute_Id } triplet identifies 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.

When the EventNotification.request service invocation does not contain the optional Application_Addresses parameter, it shall be sent from the address of the server management logical device – the default sender application – to the client management AP – the default destination application. Both APs are always present and in any protocol profile, they are bound to known, pre-defined addresses.

Use

This service is used by the server AP. Upon the reception of the EventNotification.request service invocation, the COSEM server application layer shall build the EVENT-NOTIFICATION-Request APDU.

6.6.3.3 Services provided with Short name references

6.6.3.3.1 ReadRequest

The service is described in Annex A of IEC 61334-4-41.

The parameterized access (as additional variant of the VariableAccessSpecification) provides the ReadRequest service with the capability to transport additional parameters.

Parameterized access is introduced by adding the following access method (compare IEC 61334-4-41, p. 221):

```
VariableAccessSpecification:= CHOICE
        {
                                              ...  [2]...
                                              ...  [3]...
                        parameterized access [4] IMPLICIT SEQUENCE
        {
                variable_name         ObjectName,
                selector              integer,
                parameter             Data
                }
        }
```

The meaning of the selector and of the access parameter depends on the referenced variable. It is defined in the corresponding COSEM interface class specification, see in IEC 62056-62.

6.6.3.3.2 ReadResponse

The service is described in Annex A of IEC 61334-4-41.

6.6.3.3.3 WriteRequest

The service is described in Annex A of IEC 61334-4-41.

The parameterised access (as additional variant of the VariableAccessSpecification) provides the WriteRequest service with the capability to transport additional parameters, as described above (6.6.3.3.1).

6.6.3.3.4 WriteResponse

The service is described in Annex A of IEC 61334-4-41.

6.6.3.3.5 UnconfirmedWriteRequest

The service is described in Annex A of IEC 61334-4-41.

6.6.3.3.6 InformationReportRequest

The service is described in Annex A of IEC 61334-4-41.

6.7 Summary of COSEM application layer services and service parameters

6.7.1 COSEM application layer services summary

Table 2 summarizes the COSEM application layer services on the client and the server side, and for data communication services, in case of using LN and SN referencing.

Table 2 – Application layer services – summary

	Client side	Server side
ACSE Services	COSEM-OPEN.request	COSEM-OPEN.indication
Application association establishment / release / abort	COSEM-OPEN.confirm	COSEM-OPEN.response
	COSEM-RELEASE.request	COSEM-RELEASE.indication
	COSEM-RELEASE.confirm	COSEM-RELEASE.response
	COSEM-ABORT.indication	COSEM-ABORT.indication
xDLMS Services	GET.request	GET.indication
(LN referencing)	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	
xDLMS Services	GET.request	Read.indication
(SN referencing)	GET.confirm	Read.response
	SET.request (Service_class = “confirmed”	Write.indication
	SET.confirm	Write.response
	SET.request (Service_class = “unconfirmed”	UnconfirmedWrite.indication
	ACTION.request (Service_Class = “confirmed”, with return parameters)	Read.request
	ACTION.confirm (with return parameters)	Read.response
	ACTION.request (Service_Class = “confirmed”, with no return parameters)	Write.request
	ACTION.confirm (with no return parameters)	Write.response
	ACTION.request (Service_Class = “unconfirmed”	UnconfirmedWrite.indication
	EventNotification.indication (multiple)	InformationReport.request

6.7.2 COSEM-OPEN service parameters

Table 3 summarizes the parameters of the COSEM-OPEN service in case of the various service primitives.

Table 3 – Summary of the service parameters in the COSEM-OPEN service primitives

	Client side		Server side	
	.request	.confirm	.indication	.response
Protocol_Connection_Parameters	M	M(=)	M(=)	M
Dedicated_Key	C	-	C(=)	-
Local_or_Remote	-	M	-	-
Result	-	M(=)	-	M
Failure_Type	-	C(=)	-	C
DLMS_Version_Number	M	M(=)	M(=)	M
DLMS_Conformance	M	M(=)	M(=)	M
Client_Max_Receive_PDU_Size	M	-	M(=)	-
Server_Max_Receive_PDU_Size	-	M(=)	-	M
ACSE_Protocol_Version	M	M(=)	M(=)	M
Application_Context_Name	M	M	M	M
Application_Ids_and_Titles	U	U(=)	U(=)	U
Security_Mechanism_Name	U	U(=)	U(=)	U
Calling_Authentication_Value	C	-	C(=)	
Responding_Authentication_Value	-	C(=)	-	C
Implementation_Information	U	U(=)	U(=)	U
User_Information	U	-	U	-
Service_Class	M	-	M(=)	-

The meaning of the codes used in this and the following tables is as follows:

- - Not applicable
- M Mandatory (this parameter must be present)
- U User option (this parameter may optionally be present)
- C Conditional (the presence of this parameter is conditional depends on other parameters)
- (=) Indicates that the parameter is the same as the corresponding request (for the.indication) or.response (for the.confirm) primitive.

6.7.3 COSEM-RELEASE and COSEM-ABORT service parameters

Table 4 and Table 5 summarize the parameters of the COSEM-RELEASE – and COSEM-ABORT services in case of the various service primitives.

Table 4 – Summary of the service parameters in the COSEM-RELEASE service primitives

	Client side		Server side	
	.request	.confirm	.indication	.response
Result	-	M	-	M
Failure_Type	-	C	-	-
User_Information	U	U(=)	U(=)	U
Use_RLRQ_RE	U	-	U	U

Table 5 – Summary of the service parameters in the COSEM-ABORT service primitives

	Client side		Server side	
	.request	.confirm	.indication	.response
Diagnostics	-	-	U	-

6.7.4 Parameters of the Client/Server type data communication services with LN referencing

Table 6, Table 7 and Table 8 summarize the parameters of the Client/Server type data communication services (GET, SET, ACTION) used with LN referencing, in case of the various service primitives.

Table 6 – Summary of the service parameters in the COSEM GET service primitives

	Client side		Server side	
	.request	.confirm	.indication	.response
Invoke_Id	M	M(=)	M(=)	M(=)
Priority	M	M(=)	M(=)	M(=)
Service_Class	M	-	M(=)	-
Request_Type	M	-	M(=)	-
Response_Type	-	M(=)	-	M
COSEM_Attribute_Descriptor	M	-	M(=)	-
COSEM_Class_Id	M	-	M(=)	-
COSEM_Object_Instance_Id	M	-	M(=)	-
COSEM_Object_Attribute_Id	M	-	M(=)	-
Access_Selection_Parameters	U	-	U(=)	-
Result	-	M(=)	-	M
Block_Number	C	C(=)	C(=)	C

Table 7 – Summary of the service parameters in the COSEM SET service primitives

	Client side		**Server side**	
	.request	.confirm	.indication	.response
Invoke_Id	M	M(=)	M(=)	M(=)
Priority	M	M(=)	M(=)	M(=)
Service_Class	M	-	M(=)	-
Request_Type	M	-	M(=)	-
Response_Type	-	M(=)	-	M
COSEM_Attribute_Descriptor	M	-	M(=)	-
COSEM_Class_Id	M	-	M(=)	-
COSEM_Object_Instance_Id	M	-	M(=)	-
COSEM_Object_Attribute_Id	M	-	M(=)	-
Access_Selection_Parameters	U	-	U(=)	-
Data	M	-	M(=)	-
Result	-	M(=)	-	M
Block_Number	C	C(=)	C(=)	C

Table 8 – Summary of the service parameters in the COSEM ACTION service primitives

	Client side		**Server side**	
	.request	.confirm	.indication	.response
Invoke_Id	M	M(=)	M(=)	M(=)
Priority	M	M(=)	M(=)	M(=)
Service_Class	M	-	M(=)	-
Request_Type	M	-	M(=)	-
Response_Type	-	M(=)	-	M
COSEM_Method_Descriptor	M	-	M(=)	-
COSEM_Class_Id	M	-	M(=)	-
COSEM_Object_Instance_Id	M	-	M(=)	-
COSEM_Object_Method_Id	M	-	M(=)	-
Method_Invocation_Parameters	M	-	M(=)	-
Result	-	M(=)	-	M
Response_Parameters	-	U(=)	-	U
Block_Number	C	C(=)	C(=)	C

6.7.5 EventNotification service parameters

Table 9 summarizes the parameters of the EventNotification service primitives in case of the various service primitives.

Table 9 – Summary of the service parameters in the COSEM EventNotification service primitives

	Client side	Server side
	.indication	.request
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

7 COSEM application layer protocol specification

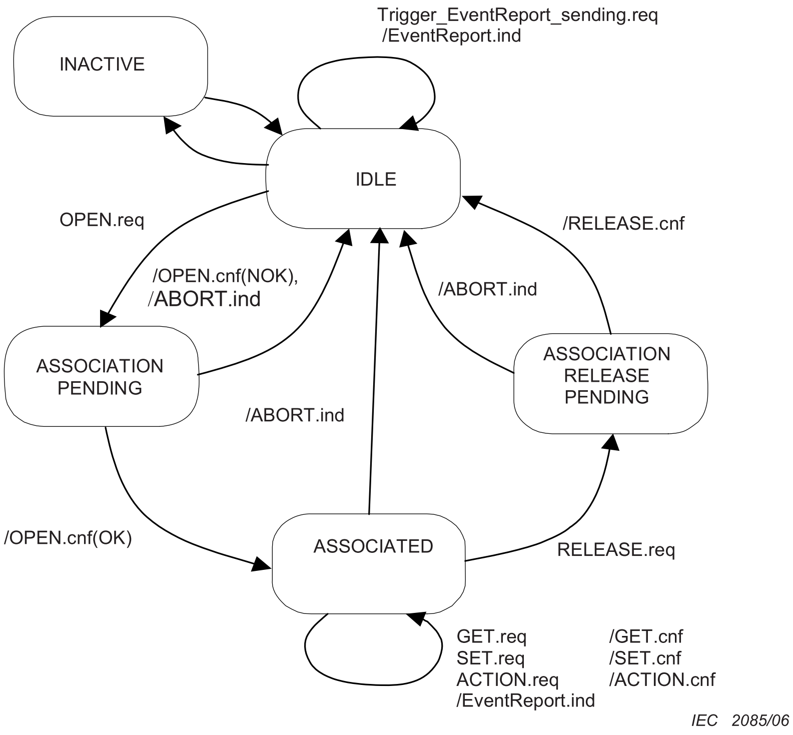
The COSEM application layer is based on the extended DLMS – xDLMS, see Annex A – and on the connection-oriented ACSE service elements. Therefore, the protocol of this layer is based on the DLMS and ACSE protocols, as they are specified in IEC 61334-4-41 and in ISO/IEC 8650-1 respectively.

Both the xDLMS and the application contexts can be negotiated during the AA establishment.

The COSEM application protocol specification includes the specification of the protocol machines for both the client and server side application layers, and the abstract syntax (ASN.1) for the representation of APDUs. As the same APDU applies at the client side and at the server side, for example a.request type APDU, sent by the client is the same as its peer.indication APDU, the abstract syntax specification is common for both application layer entities and is given in Clause 8.

7.1 State definitions for the client side control function

Figure 16 shows the state machine for the client side control function (CF, see Figure 5).



NOTE On the state diagrams, the following conventions are used:

– service primitives with no "/" character as first character are "stimulants": the invocation of these services is the origin of the given state transition;

– service primitives with an "/" character as first character are "outputs": the invocation of these services is done on the state transition path.

Figure 16 – Partial state machine for the client side control function

Definitions of states are as follows:

- INACTIVE – in this state, the client CF (and the application layer) 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 of the client application layer protocol entity when there is no AA created, being released or currently established[9]. Nevertheless, some data exchange between the client and server, if the physical channel is already established, is possible in this state.

State transitions between the INACTIVE and IDLE states are controlled outside of the protocol. For example, it can be considered that the CF, and with it the application layer including it, makes the state transition from INACTIVE to IDLE state by being instantiated and bound on the top of the supporting protocol layer. The opposite transition may happen by deleting the given instance of the CF (application layer).

- ASSOCIATION PENDING – the CF of the application layer entity enters this state when the COSEM client AP invokes the COSEM-OPEN.request (OPEN.req) service primitive. The CF may exit from this state either by sending a COSEM-OPEN.confirmation (/OPEN.cnf) service primitive or, in the case of physical disconnection, by sending a COSEM-ABORT.indication (/ABORT.indication) service primitive to the AP. Depending on the result of the association request, the client CF shall return to IDLE state (NOK), or shall enter the ASSOCIATED state;

[9] Note, that it is the state machine for the application layer: lower layer connections, including the physical connection, are not taken into account. On the other hand, physical connection establishment is done outside the protocol.

- ASSOCIATED – the CF shall enter this state when the AA has been successfully established. Data communication services – GET, SET, ACTION – are provided only in this state. The client CF shall remain in this state until the AP explicitly requires the release of the association by invoking the COSEM-RELEASE.request service primitive (RELEASE.req), or a COSEM-ABORT.indication service is invoked;
- ASSOCIATION RELEASE PENDING – the CF of the application layer entity enters this state when the COSEM client AP invokes the COSEM-RELEASE.request service primitive (RELEASE.req), requesting the release of the established AA. The CF shall remain in this state, waiting for the response to this request. As the server is not allowed to refuse a release request, after exiting this state, the CF shall always enter the IDLE state. The exit from this state can be originated either by the reception of a COSEM-RELEASE.response from the remote server, the local generation of the COSEM-RELEASE.confirm or by the invocation of a COSEM-ABORT.indication service primitive.

7.2 State definitions for the server side control function

Figure 17 shows the state machine for the server side control function, see Figure 5.

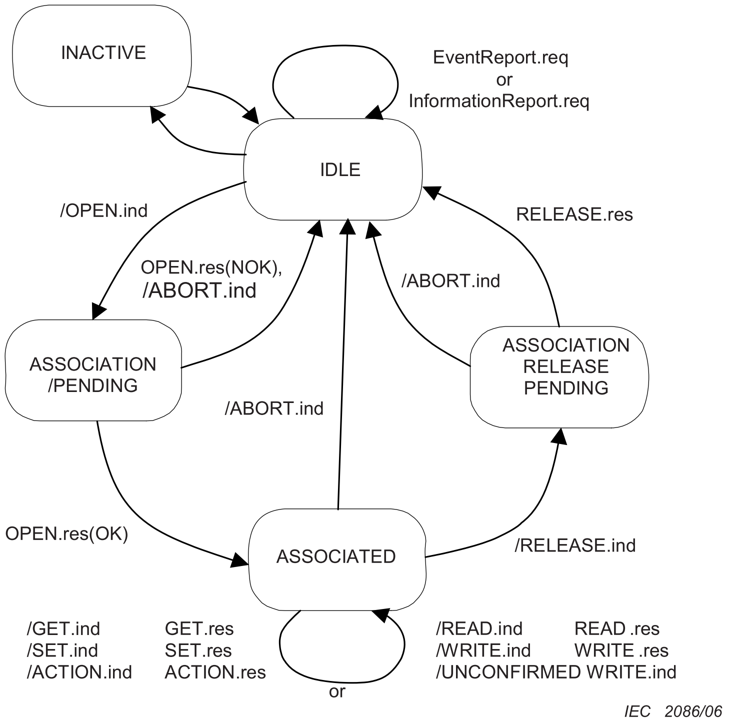


Figure 17 – Partial state machine for the server side control function

Definitions of the states are as follows:

- INACTIVE – in this state, the server CF (and the application layer) 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 of the server application layer entity when there is no AA created, being released or currently established. Nevertheless, some data exchange between the client and server, if the physical channel is already established, is possible in this state;
- ASSOCIATION PENDING – upon the reception of a COSEM-OPEN.request message from a remote client, the CF of the server application layer protocol entity shall exit the IDLE state. It shall indicate the reception of this message to the server AP via the COSEM-OPEN.indication service primitive (/OPEN.indication) and shall enter into ASSOCIATION PENDING state. In this state, the Server CF is waiting for the response from the AP. If the response is positive – meaning that the AP accepted the proposed association – the CF

shall enter the ASSOCIATED state. If the response is negative – or if a physical disconnection is detected – the CF shall return to the IDLE state;

- ASSOCIATED – the server CF shall enter this state when the AA has been successfully established. Data communication services – GET, SET, ACTION or READ, WRITE and UNCONFIRMED WRITE, depending on the established application context – are provided only in this state. The server CF shall remain in this state until the remote client explicitly requires the release of the AA by invoking the COSEM-RELEASE.request service (/RELEASE.ind), or a COSEM-ABORT.indication service is invoked;
- ASSOCIATION RELEASE PENDING – upon the reception of a COSEM-RELEASE.request service primitive from the remote client AP, the CF of the application layer protocol entity shall indicate it to the AP (/RELEASE.indication) and shall enter into this state. The CF shall remain in this state, waiting for the response invocation from the AP. As the server is not allowed to refuse this request, the CF shall always enter the IDLE state after leaving the ASSOCIATION RELEASE PENDING state. The exit from this state can be originated also by the invocation of a COSEM-ABORT.indication service primitive.

7.3 Protocol for application association establishment/release

7.3.1 Establishment of a confirmed application association

Application association establishment with the help of the Association.request/.indication./.response/.confirmation services of the standard ACSE, ISO/IEC 8650-1, is the key element of COSEM interoperability. The participants of an AA are the interoperable communications partners:

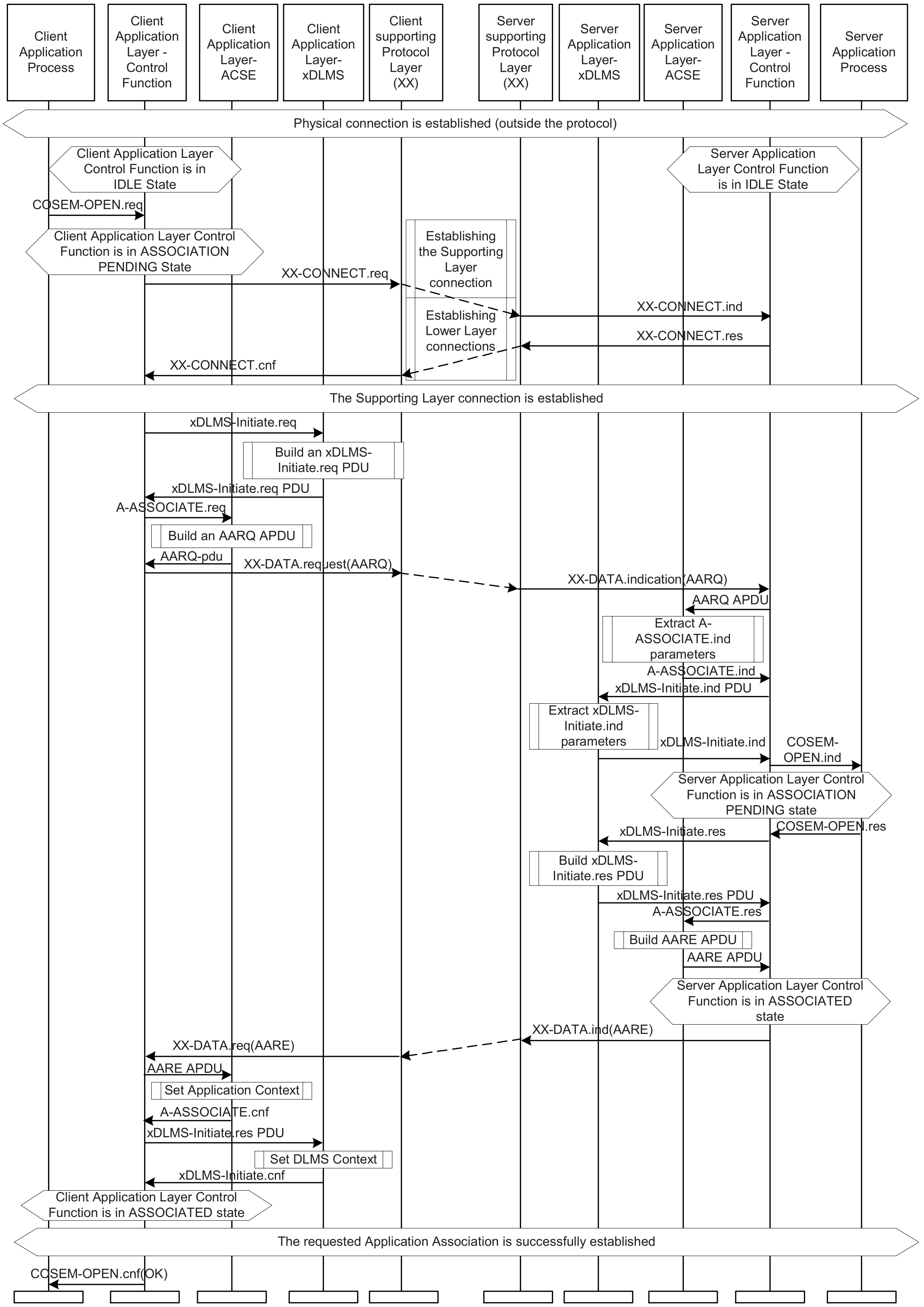
- a client AP, which is always the originator of an AA request, and
- a server AP[10].

The client AP shall first invoke the COSEM-OPEN.request service of the client COSEM ASO. Upon the reception of this service invocation, the Control function of the client ASO shall first examine whether the establishment of a lower layer connection is required for the requested AA or not. In this case, it shall first establish the required lower layer connection(s).

Figure 18 gives the MSC for the case, when:

- the COSEM-OPEN.request service is requesting for a confirmed AA;
- the connection of the supporting lower layers is required for the establishment of the required AA.

[10] In order to be able to provide multicast and broadcast services, in COSEM an AA can also be established between a client and a group of server application processes.


Client Application Process
Client Application Layer - Control Function
Client Application Layer- ACSE
Client Application Layer- xDLMS
Client supporting Protocol Layer (XX)
Server supporting Protocol Layer (XX)
Server Application Layer- xDLMS
Server Application Layer- ACSE
Server Application Layer - Control Function
Server Application Process
Physical connection is established (outside the protocol)
Client Application Layer Control Function is in IDLE State
Server Application Layer Control Function is in IDLE State
COSEM-OPEN.req
Client Application Layer Control Function is in ASSOCIATION PENDING State
Establishing the Supporting Layer connection
XX-CONNECT.req
Establishing Lower Layer connections
XX-CONNECT.ind
XX-CONNECT.res
XX-CONNECT.cnf
The Supporting Layer connection is established
xDLMS-Initiate.req
Build an xDLMS-Initiate.req PDU
xDLMS-Initiate.req PDU
A-ASSOCIATE.req
Build an AARQ APDU
AARQ-pdu
XX-DATA.request(AARQ)
XX-DATA.indication(AARQ)
AARQ APDU
Extract A-ASSOCIATE.ind parameters
A-ASSOCIATE.ind
xDLMS-Initiate.ind PDU
Extract xDLMS-Initiate.ind parameters
xDLMS-Initiate.ind
COSEM-OPEN.ind
Server Application Layer Control Function is in ASSOCIATION PENDING state
COSEM-OPEN.res
xDLMS-Initiate.res
Build xDLMS-Initiate.res PDU
xDLMS-Initiate.res PDU
A-ASSOCIATE.res
Build AARE APDU
AARE APDU
Server Application Layer Control Function is in ASSOCIATED state
XX-DATA.ind(AARE)
XX-DATA.req(AARE)
AARE APDU
Set Application Context
A-ASSOCIATE.cnf
xDLMS-Initiate.res PDU
Set DLMS Context
xDLMS-Initiate.cnf
Client Application Layer Control Function is in ASSOCIATED state
The requested Application Association is successfully established
COSEM-OPEN.cnf(OK)

IEC 2087/06

Figure 18 – MSC for successful application association establishment preceded by a successful lower layer connection establishment

Once the required lower layer connections are established, the client CF shall assemble an AARQ APDU with the help of the two application service elements (ACSE and xDLMS) of the client application layer. This AARQ APDU shall be the first message sent to the server application layer.

The CF of the server application layer shall first give the received AARQ APDU to the ACSE, which shall extract the ACSE related parameters, then give back the control to the CF. The CF shall send the contents of the user-information field of the AARQ APDU to the xDLMS-ASE, as a xDLMS-Initiate.indication DLMS PDU.

The xDLMS-ASE shall retrieve the parameters of the xDLMS-Initiate.indication. It shall then give back the control to the CF, which shall invoke the COSEM-OPEN.indication service primitive with the appropriate parameters, extracted from the AARQ APDU[11], to the COSEM server AP. At the same time, the server Control function shall enter the 'ASSOCIATION PENDING' state.

The server AP shall analyze the received COSEM-OPEN.indication primitive, and decide whether it accepts the proposed AAs or not[12].

Following this verification, and if the proposed AA is confirmed, the COSEM server AP shall invoke the COSEM-OPEN.response service to indicate the acceptance or non-acceptance of the proposed association. The CF shall assemble and send the appropriate AARE APDU to the remote peer client application layer via the supporting lower protocols. If the requested AA is non-confirmed, no AARE is sent. If the proposed AA has been accepted, the server is able to receive xDLMS data communication service.request(s) and to send.responses to confirmed service requests within this AA. In other words, the association has been established, and the server has entered the data communications phase.

At the client side, the parameters of the received AARE APDU shall be extracted by the help of the ACSE and the xDLMS-ASE, and shall be sent to the client AP via the COSEM-OPEN.confirm service primitive. At the same time, the client application layer shall enter the 'ASSOCIATED' state. From this moment, the AA is established within the negotiated application and xDLMS contexts.

7.3.2 Establishment of special application associations

7.3.2.1 Pre-established application associations

Pre-established AAs do not need to be established using the COSEM-OPEN service. It can be considered, that this OPEN has already been done (it does not matter how). Consequently, pre-established AAs can be considered existing from the moment the lower layers are able to deliver APDUs between the client and the server.

This standard does not specify the way of establishing these associations.

7.3.2.2 Establishment of non-confirmed application associations

Invoking the COSEM-OPEN.request service with Service_class == Unconfirmed shall result in the establishment of a non-confirmed AA. In this case, the AARQ APDU shall be sent with the response-allowed field of the included xDLMS-Initiate.Request PDU set to FALSE – meaning, that no AARE is expected.

[11] Some service parameters of this 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. The Service_Class parameter of the COSEM-OPEN service is linked to the response-allowed field of the xDLMS-Initiate.request APDU.

[12] The application service elements only extract the parameters, like the application context, authentication related parameters, etc. The interpretation of these parameters and the decision whether the association can be accepted or not, is the job of the COSEM server application process.

Once a non-confirmed AA is established, the client COSEM application layer shall accept only non-confirmed xDLMS service requests (GET, SET, ACTION) within this AA. The main purpose of having this type of AA is to allow multi-casting and broadcasting.

A non-confirmed COSEM-OPEN.request is always locally confirmed.

7.3.3 The AARQ and AARE APDUs

The standard connection-oriented ACSE provides several functional units in order to negotiate ACSE user requirements during association establishment. In COSEM, only two of them are used: the kernel and the authentication functional units.

The kernel functional unit is always available – it is the default functional unit. The authentication functional unit is present only when it is explicitly requested[13]. The selection of the authentication functional unit supports additional fields on the AARQ and AARE APDUs.

The AARQ and AARE APDUs specifications are as follows:

```
AARQ-apdu::= [APPLICATION 0] IMPLICIT SEQUENCE
  {
        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] IMPLICIT association-information OPTIONAL
  }
```

and

```
AARE-apdu::= [APPLICATION 1] IMPLICIT SEQUENCE
  {
        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] IMPLICIT association-information OPTIONAL
  }
```

[13] The presence of this functional unit – and the optional fields corresponding to the usage of this functional unit – depend on the authentication security level.

The values of the AARQ and AARE fields in COSEM are the following:

- **protocol-version**: the ACSE protocol-version. The default value is used;
- **application-context-name:** the appropriate COSEM application-context-name is used. Application-context-names for the default COSEM application contexts are specified in 7.3.7.1;
- OPTIONAL called, calling and responding titles, qualifiers and identifiers: These optional fields carry the contents of the optional Application_Ids_and_Titles parameter of the COSEM_OPEN service. The usage of these fields is as it is specified in the ACSE standard ISO/IEC 8649;

  If these fields are present in the AARQ, but the server is not able to recognize them, then it may ignore them. In this case, these parameters shall not influence the association establishment and the AARE shall not contain any of these fields. On the other hand, if the server recognizes these parameters, it shall take into account the value of these parameters to establish the AA, and these fields shall be present in the AARE;
- **sender and responder acse requirements:** when present, it carries the value of BIT STRING { authentication (0) };
- **mechanism-name:** when present, it contains the name of the authentication mechanism. COSEM authentication mechanism names are specified in 7.3.7.2;

  NOTE In the AARQ, the mechanism name defines the authentication mechanism required by the client, i.e. the authentication mechanism, which is expected to be used by the server. In the AARE, the mechanism name defines the authentication mechanism required by the server, i.e. the mechanism, which should be used by the client.
- **calling and responding authentication value:** when present, it is specific to implementation and is not within the scope for this standard;
- **implementation-information:** the usage of this field is based on prior agreement between the communicating stations. This usage is not defined in this standard;
- **user-information:** this parameter shall always be present and shall contain an – optionally encrypted – xDLMS-Initiate.request PDU in the case of AARQ APDU and a xDLMS-Initiate.response PDU or DLMS-ConfirmedServiceError PDU in the case of an AARE APDU;
- **result**: this parameter carries the result of the proposed AA establishment;
- **result-source-diagnostics:** this field carries the result and eventually the reason of the rejection of the association establishment request, as it is specified in ISO/IEC 8650-1. When no diagnostics are included, a null value is assigned to the result-source-diagnostics field.

Both the AARQ and the AARE APDUs encoded in BER (ISO/IEC 8825). On the other hand, the user-information field of these APDUs, carrying the xDLMS-Initiate.request/.response (or ConfirmedServiceError) DLMS PDU-s shall be encoded in A-XDR, see IEC 61334-6 Examples for AARQ/AARE APDU encoding are given in Annex C.

7.3.4 Managing the parameters for application association establishment

According to the protocol described above, an AA establishment is proposed by the client, and accepted or not accepted by the server. The conditions under which the server accepts or rejects the establishment of an AA are defined in the following subclauses.

There are two contexts negotiated via the COSEM-OPEN service: the COSEM application context and the xDLMS context. The elements of the COSEM application context are carried by the fields of the AARQ APDU. The xDLMS context is defined by the parameters of the xDLMS-Initiate.request/.response PDUs, carried by the user-information field of the AARQ/AARE. See also Annex A.

Upon the receipt of the AARQ APDU, the server shall first check the COSEM application context. If the proposed application context is not acceptable, the proposed AA shall be refused. (e.g. application context name is different, the authentication mechanism name and authentication value are expected but not provided, the authentication value is not correct, etc.).

The parsing order of the AARQ and AARE shall be the following:

a) lower layer parameters;

   NOTE These parameters are not carried by the AARQ/AARE, but they are provided by the supporting layer.

b) AARQ syntax;

c) ACSE protocol version;

d) application context name;

e) authentication related fields:

   - if **sender ACSE requirements** is present but bit 0=0 or if the parameter is not present: any following authentication parameters may be ignored;
   - if **sender ACSE requirements** is present and bit 0=1, and the following authentication parameters are inconsistent, then an error message shall be returned.

When the parsing of the above AARQ fields is completed, the server shall continue parsing the parameters of the xDLMS-Initiate.request PDU as follows:

f) response-allowed:

   If the proposed association is confirmed, the value of this parameter is TRUE, the server shall send back an AARE. Otherwise, the value is FALSE, the server shall not respond. The default value is TRUE;

g) proposed-dlms-version-number;

h) proposed-conformance;

i) client-max-receive-pdu-size.

If the proposed application context is not acceptable for the server, it shall refuse the proposed AA with the reason of non-fit at the COSEM application context level, and if the value of the response-allowed parameter is TRUE, it shall send back an AARE APDU, containing diagnostics information about reasons for the failure. The application-context-name field in the AARE shall be the same as in the AARQ. Optionally, the value of the application context-name-field in the AARE may be different from the value in the AARQ. In this way, the server is able to indicate (one of) the application context(s) it supports. The user-information field of the response AARE APDU shall contain the xDLMS context supported by the server: the supported DLMS version number, the supported conformance block and the server-max-receive-pdu-size.

If both the proposed application context and the xDLMS context are acceptable, the server shall establish the proposed AA. If the value of the response-allowed parameter is TRUE, the server shall send back an AARE APDU, containing the indication of the success and a correctly constructed xDLMS-Initiate.response PDU in the user-information field. This shall carry the parameters of the negotiated xDLMS context. The value of the negotiated-conformance field of this PDU shall always be the negotiated conformance block: a bit per bit AND of the received conformance block and the server's own conformance block. See 8.5.

If the xDLMS context proposed by the client cannot be accepted, the server shall refuse the proposed association. In this case – application context fits, but xDLMS context does not fit (e.g. the value of the negotiated conformance block is zero) – the server shall send back an AARE APDU[14], with "no-reason-given" as diagnostics information. The user-information field

[14] Only if the proposed AA is confirmed.

of this AARE shall contain a correctly constructed DLMS-ConfirmedServiceError message, indicating the reason for the failure.

7.3.5 Repeated COSEM-OPEN.request service invocations

The reception of a COSEM-OPEN.request service invocation referencing to an already existing AA will be locally and negatively confirmed by the Client application layer, and has no effect on the already established AA.

If, nevertheless, a server application layer receives an AARQ APDU referencing to an already existing AA, it will simply discard this AARQ, or, if it is implemented, it may also respond with the optional EXCEPTION-Response APDU.

7.3.6 Releasing an application association

7.3.6.1 Overview

An existing AA can be released gracefully or non-gracefully. Graceful release means that it is the protocol machine, which notifies its peer that it is releasing the association. Graceful release can be initiated only by the client AP.

Non-graceful release means that the association is unexpectedly terminated. The reason for such an event is always outside of the protocol: it can be, for example the detection of a physical disconnection not initiated by the AP.

7.3.6.2 Graceful release of an application association

Graceful release of an AA is always initiated by the client AP by invoking the COSEM-RELEASE.request service.

According to the protocol of the application layer ACSE, on the receipt of the COSEM-RELEASE.request, the COSEM Client ASO invokes the A_RELEASE.request service of the ACSE, which results then sending out an RLRQ APDU to the server, indicating that a graceful release of the AA is requested.

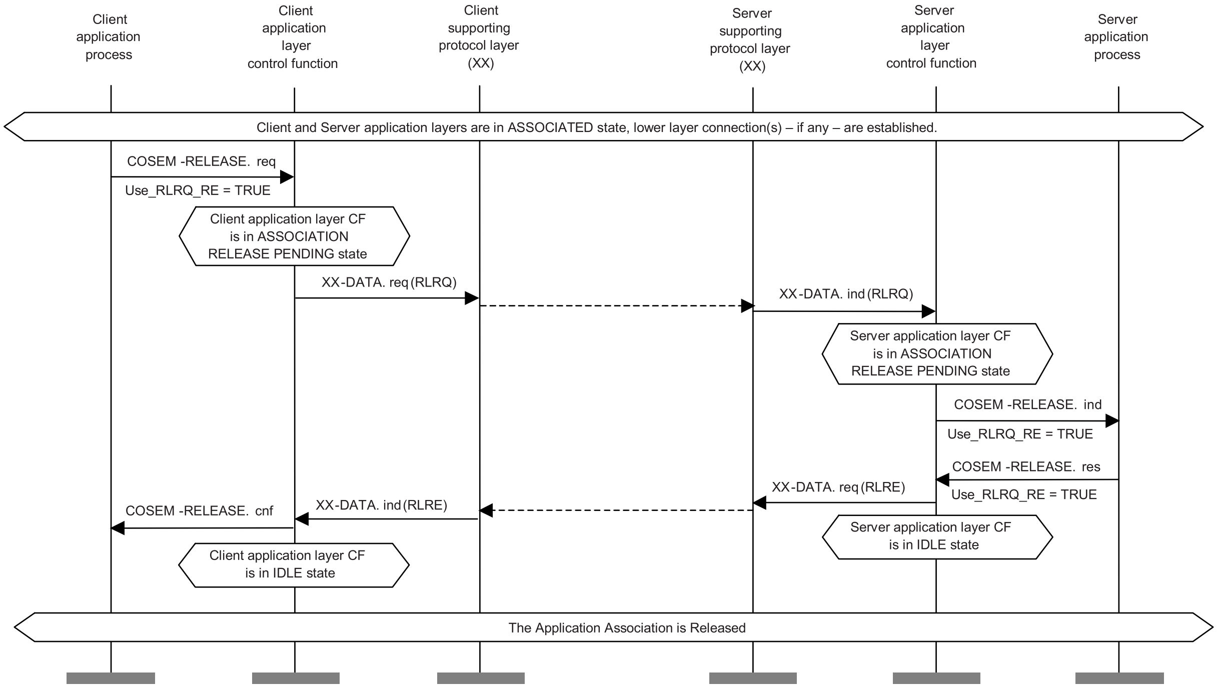
The COSEM application layer shall optionally support this operation. When the COSEM-RELEASE.request service is invoked with Use_RLRQ_RE == TRUE, the AA shall be released with the help of the Association Release services of the ACSE.

In communication profiles, where the connection and disconnection of the supporting layer is not managed by the COSEM application layer, or where the supporting layer is connectionless, releasing AA-s using the ACSE A-RELEASE services is the only possible way, thus it is mandatory. See also Clause B.3.

Using the A-RELEASE services is also the only possibility to remotely release non-confirmed AA-s.

NOTE If the ACSE A_RELEASE services are not supported, then non-confirmed AA-s can be locally released on the client side. On the server side, they can be released by using a time out or by any other suitable mechanisms.

An example for releasing AAs using the ACSE A-RELEASE services is shown in Figure 19.


Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and Server application layers are in ASSOCIATED state, lower layer connection(s) – if any – are established.
COSEM -RELEASE. req
Use_RLRQ_RE = TRUE
Client application layer CF is in ASSOCIATION RELEASE PENDING state
XX-DATA. req (RLRQ)
XX-DATA. ind (RLRQ)
Server application layer CF is in ASSOCIATION RELEASE PENDING state
COSEM -RELEASE. ind
Use_RLRQ_RE = TRUE
COSEM -RELEASE. res
Use_RLRQ_RE = TRUE
XX-DATA. req (RLRE)
XX-DATA. ind (RLRE)
COSEM -RELEASE. cnf
Server application layer CF is in IDLE state
Client application layer CF is in IDLE state
The Application Association is Released

IEC 2088/06

Figure 19 – Graceful association release using the A-RELEASE service

In certain COSEM communication profiles – for example in the 3-layer, connection-oriented, HDLC based profile, see Clause B.2 – there is a one-to-one relationship between a confirmed AA and the supporting data link layer connection. In this case, a simpler mechanism is available: confirmed AAs can be released simply by disconnecting the corresponding lower layer connection.

In any COSEM communication profile, disconnecting the supporting layer shall release all AAs built on that supporting layer connection.

Figure 20 gives an example of a confirmed AA graceful releasing by disconnecting the corresponding lower layer connection.

Invocation of the COSEM-RELEASE.request service with no Use_RLRQ_RE or with Use_RLRQ_RE == FALSE shall imply directly the invocation of an XX-DISCONNECT.request service primitive. This request shall initiate the disconnection of the lower protocol layers. As a result of the required message exchanges at the lower protocol layer level, the disconnect request shall be indicated to the server application layer via the XX-DISCONNECT.indication service primitive.

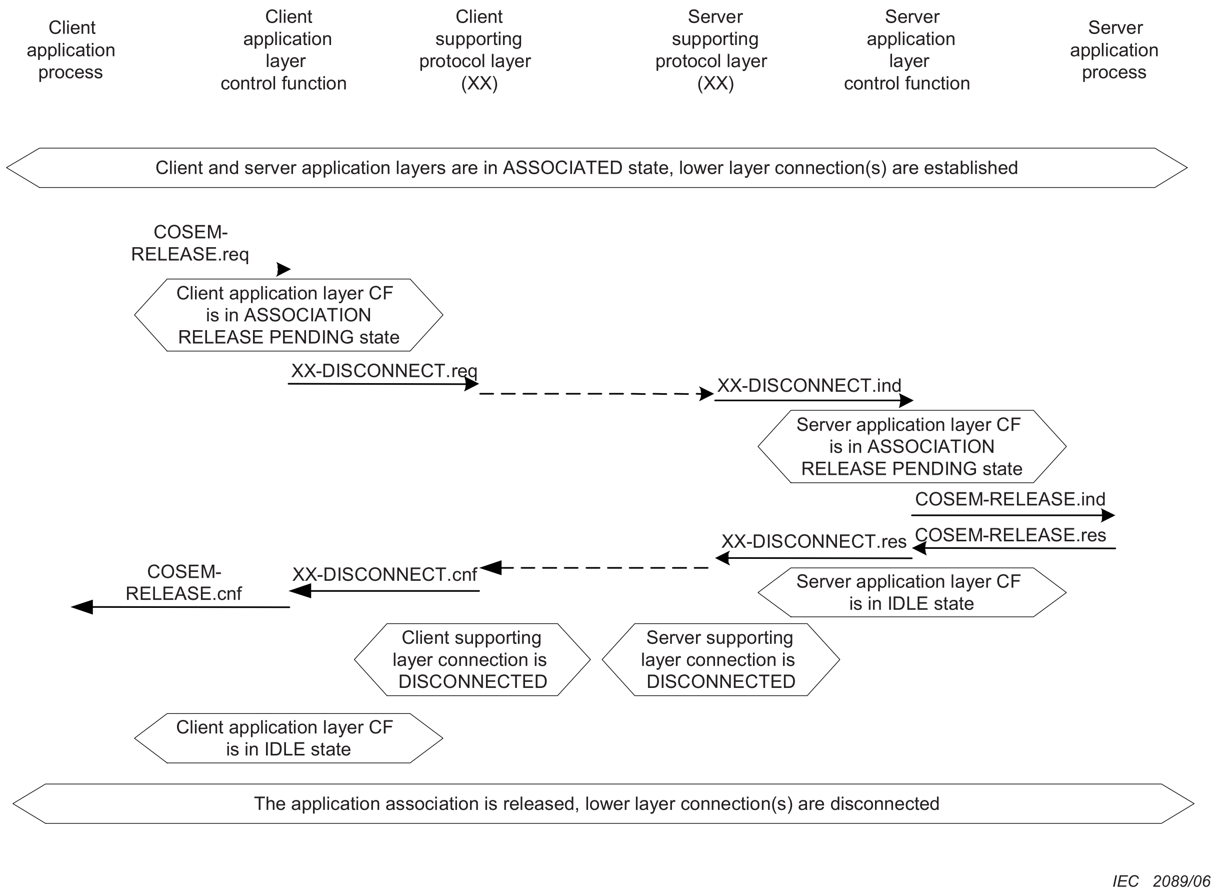

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s) are established
COSEM-RELEASE.req
Client application layer CF is in ASSOCIATION RELEASE PENDING state
XX-DISCONNECT.req
XX-DISCONNECT.ind
Server application layer CF is in ASSOCIATION RELEASE PENDING state
COSEM-RELEASE.ind
XX-DISCONNECT.res
COSEM-RELEASE.res
COSEM-RELEASE.cnf
XX-DISCONNECT.cnf
Server application layer CF is in IDLE state
Client supporting layer connection is DISCONNECTED
Server supporting layer connection is DISCONNECTED
Client application layer CF is in IDLE state
The application association is released, lower layer connection(s) are disconnected
IEC 2089/06

Figure 20 – Graceful release of an application association by disconnection of the supporting layer

The server application layer shall interpret this XX-DISCONNECT.indication as a request for releasing the AA, and shall indicate this request to the COSEM server AP via the COSEM-RELEASE.indication service primitive.

The COSEM server AP shall accept the required disconnection and shall invoke the COSEM-RELEASE.response service with the appropriate parameters.

Upon the receipt of the COSEM-RELEASE.response service invocation, the server application layer shall invoke the XX-DISCONNECT.response service of the supporting protocol layer with the appropriate parameters. At the same moment, the Control function of the server application layer shall enter the ‘IDLE’ state[15.]

Invocation of the XX-DISCONNECT.response service primitive causes the server supporting layer(s) to disconnect the related connection(s) and to inform about it the peer supporting layer(s). The reception of this information shall be indicated to the client application layer by the XX-DISCONNECT.confirm primitive, which is relayed to the client AP by the client application layer via the COSEM-RELEASE.confirm service primitive. The invocation of this primitive means that the association has been successfully released.

[15] The release of the existing application association may require internal communication among the application service elements (ACSE, xDLMS-ASE) and the Control function inside the server application layer. These interactions are not shown in the figures.

7.3.6.3 Non-graceful release of an application association

Various events may result in 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.indication service. The Diagnostics parameter of this service indicates the reason for the non-graceful AA release.

Figure 21 shows the message sequence chart for aborting the AA, due to the detection of a physical disconnection.

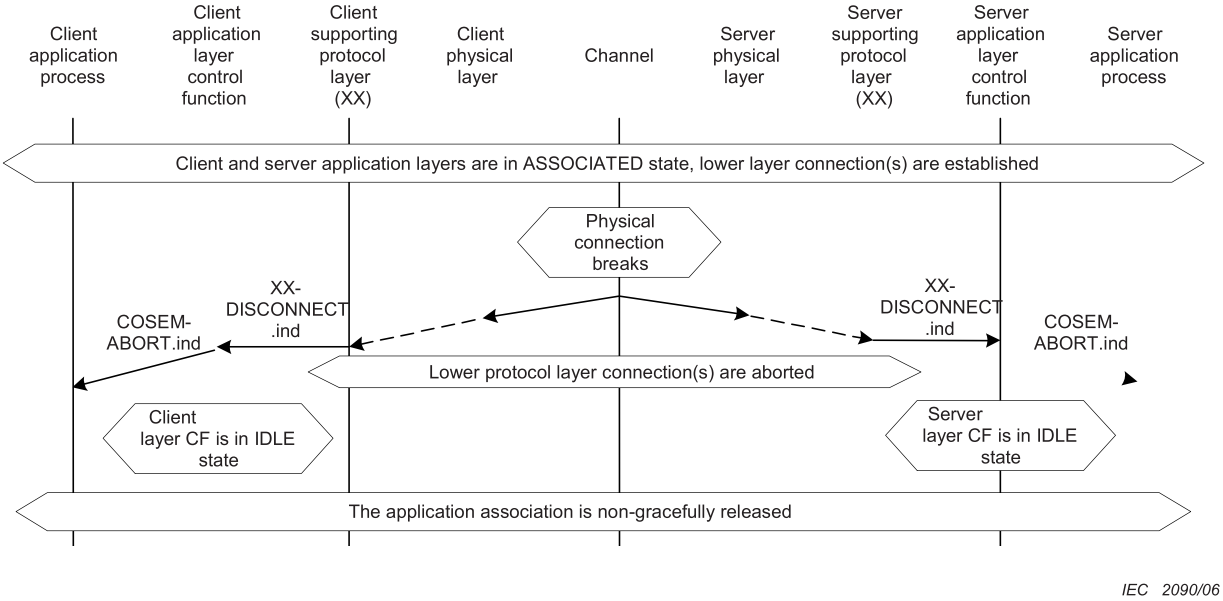

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Client physical layer
Channel
Server physical layer
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s) are established
Physical connection breaks
XX-DISCONNECT .ind
COSEM-ABORT.ind
XX-DISCONNECT .ind
COSEM-ABORT.ind
Lower protocol layer connection(s) are aborted
Client layer CF is in IDLE state
Server layer CF is in IDLE state
The application association is non-gracefully released
IEC 2090/06

Figure 21 – Aborting an application association following a PH-ABORT.indication

The non-graceful release of AA is not selective: if it happens, all the existing association(s) shall be aborted.

7.3.7 Registered COSEM names

Within an OSI environment, many different types of network objects must be identified with globally unambiguous names. These network objects include abstract syntaxes, transfer syntaxes, application contexts, authentication mechanism names, etc. Names for these objects in most cases are assigned by the committee developing the particular basic ISO standard or by implementers' workshops, and should be registered. For the COSEM environment, these objects are assigned by the DLMS User Association, and are specified in this standard.

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

- COSEM application context names (for LN and SN references, without or with cyphering);
- COSEM authentication mechanism names.

7.3.7.1 The COSEM application context

In order to effectively exchange information within an AA, the pair of AE-invocations shall be mutually aware of, and follow a common set of rules that govern the exchange. This common set of rules is called the application context of the AA.

The application context that applies to an AA is determined during its establishment[16]. 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)}

where the value of the context_id parameter selects a pre-existing application context.

There are four application context names specified:

COSEM_Application_Context_Name-Logical_Name_Referencing_no_ciphering::= {joint-iso-ccitt(2) country(16) country-name(756) identified-organisation(5) DLMS-UA(8) application-context(1) context_id(1)}
COSEM_Application_Context_Name-Short_Name_Referencing_no_ciphering::= {joint-iso-ccitt(2) country(16) country-name(756) identified-organisation(5) DLMS-UA(8) application-context(1) context_id(2)}
COSEM_Application_Context_Name-Logical_Name _with_ciphering::= {joint-iso-ccitt(2) country(16) country-name(756) identified-organisation(5) DLMS-UA(8) application-context(1) context_id(3)}
COSEM_Application_Context_Name-Short_Name_Referencing_with_ciphering::= {joint-iso-ccitt(2) country(16) country-name(756) identified-organisation(5) DLMS-UA(8) application-context(1) context_id(4)}

The meaning of these COSEM application contexts is:

- there are two ASEs present within the application-entity invocation, the ACSE and the xDLMS-ASE;
- the xDLSM-ASE is as it is specified in 61134-4-41[17];
- the transfer syntax is A-XDR;
- context_id(1): logical name referencing, no ciphering used;
- context_id(2): short name referencing, no ciphering used;
- context_id(3): logical name referencing, ciphering used;
- context_id(4): short name referencing, ciphering used.

NOTE Ciphering algorithms are not defined in this standard.

[16] 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.

[17] With the COSEM extensions to DLMS, see Annex A.

In order to successfully establish an AA, the AARQ and AARE APDUs should carry one of the “valid” values in their application-context-name fields. However, when the server does not accept the proposed application context, the application-context-name field in the AARE response may optionally be different from the application-context-name received in the AARQ. The server may indicate in this way to the client one of the application-context-names supported by the server.

7.3.7.2 COSEM authentication mechanism names

Authentication is one of the security aspects addressed by the COSEM specification. In order to provide different levels of security for authentication support, COSEM specifies three levels of authentication securities:

- 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 uses the authentication feature of the connection-oriented ACSE and for high level authentication, also the methods of the Association LN/SN objects. The process of LLS and HLS authentication is described in IEC 62056-62. To identify the authentication mechanism used, the following object identifiers for authentication mechanism names are specified:

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 parameter selects one of the specified security mechanisms. There are five authentication mechanism names specified:

default-COSEM-lowest-level-security-mechanism-name[18]::=

{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) authentication_mechanism_name(2) mechanism_id(0)}

default-COSEM-low-level-security-mechanism-name::=

{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) authentication_mechanism_name(2) mechanism_id(1)}

default-COSEM-high-level-security-mechanism-name::=

{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) authentication_mechanism_name(2) mechanism_id(2)}

NOTE The mechanism name for high-level security starts from 2 and it is registered by the DLMS UA. Mechanism_id(2) is manufacturer specific.

default-COSEM-high-level-security-mechanism-name_using_MD5::=

{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) authentication_mechanism_name(2) mechanism_id(3)}

default-COSEM-high-level-security-mechanism-name_using_SHA-1::=

{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) authentication_mechanism_name(2) mechanism_id(4)}

The mechanism name element of the AARQ/AARE APDU is present only, when authentication is used. See 7.3.3.

[18] This mechanism is used for the mandatory AA between a public client and the management logical device in a physical metering device.

7.4 Protocol for data communications

All data communication services are client/server services, except the EventNotification services. Data communication is always initiated by the client by invocation of GET/SET/ACTION.request services. Upon invocation of any of these services, the client application layer protocol machine builds the corresponding APDU and sends it to the peer server application layer.

Data communication service requests can be invoked in a confirmed or an unconfirmed manner. When a service is invoked in a confirmed manner, the server shall respond to the request, otherwise no application level confirmation is expected. See 7.4.1.1.

Unconfirmed services might be invoked in two different ways: individually addressed or broadcast (multicast). See 7.4.1.2.

There is a fourth, non-client/server data communications service supported, the Event Notification service. By requesting this service, the server AP is able to send an unsolicited notification of the occurrence of an event to the remote client. See 7.4.1.3.

7.4.1 Protocol for the xDLMS services using LN referencing

7.4.1.1 Protocol for confirmed services

For confirmed data communication services, the following service primitives are available:

- GET (.request/.indication/.response/.confirm);
- SET (.request/.indication/.response/.confirm);
- ACTION (.request/.indication/.response/.confirm).

GET and SET services are referencing attribute(s) of COSEM interface object instances. The ACTION service is referencing a method of a COSEM interface object instance (e.g. capture a pre-defined set of data). For definition of attributes and methods of COSEM interface classes, see IEC 62056-62.

The COSEM client may invoke the.request primitive of these services in a confirmed manner within a confirmed AA only.

The COSEM server AP, upon the receipt of a data communication service indication, shall check whether the service can be provided or not (validity, client access rights, availability, etc.). If everything is OK, it locally applies the required service on the corresponding ‘real’ object. If a response is required, the COSEM server AP shall generate the appropriate.response message.

Figure 22 shows a complete message sequence chart for a confirmed GET.request service invocation in case of success.

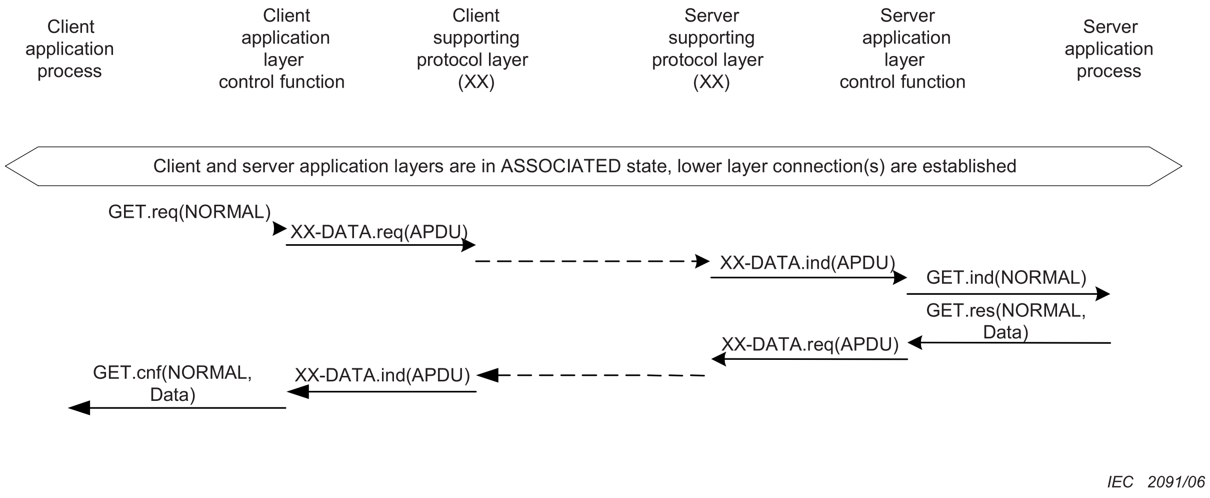

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s) are established
GET.req(NORMAL)
XX-DATA.req(APDU)
XX-DATA.ind(APDU)
GET.ind(NORMAL)
GET.res(NORMAL, Data)
XX-DATA.req(APDU)
XX-DATA.ind(APDU)
GET.cnf(NORMAL, Data)
IEC 2091/06

Figure 22 – MSC for a confirmed GET service in case of success

NOTE The message sequence on the figure above applies only if the transferred data does not exceed the supported maximum size of the APDU. In order to be able to transfer longer data with the GET service, COSEM provides an application layer level protocol. In addition, a data link layer level protocol is also available, which is transparent for the application layer. See 7.4.1.8.1.

Figure 23 shows the complete message sequence chart for a confirmed SET service, in case of success.

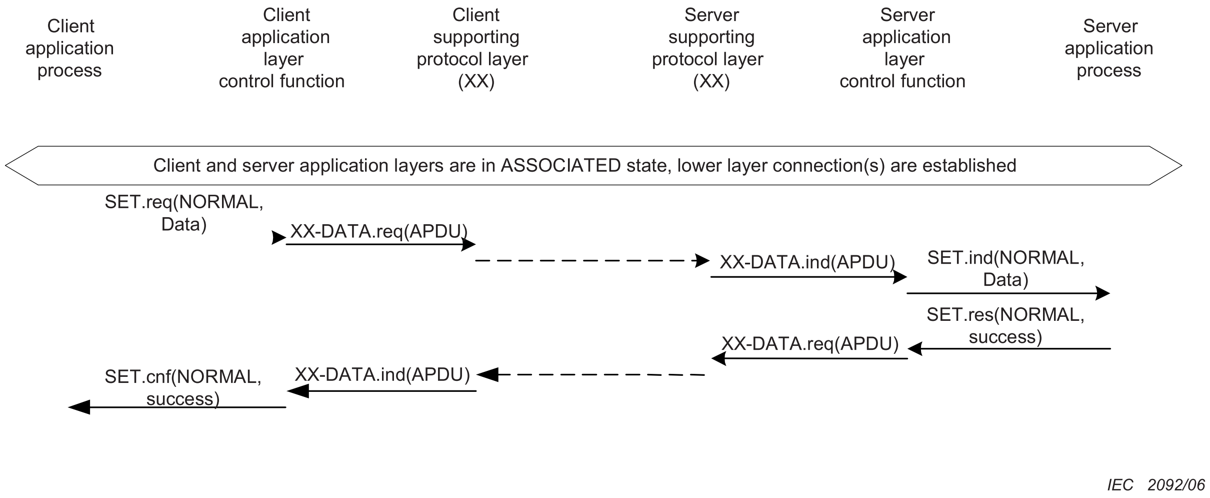

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s) are established
SET.req(NORMAL, Data)
XX-DATA.req(APDU)
XX-DATA.ind(APDU)
SET.ind(NORMAL, Data)
SET.res(NORMAL, success)
XX-DATA.req(APDU)
XX-DATA.ind(APDU)
SET.cnf(NORMAL, success)
IEC 2092/06

Figure 23 – MSC for a confirmed SET service in case of success

In case of failure, the server – instead of a positive acknowledgement, shown on the above figure – shall send a negative acknowledgement, indicating the reason for the failure, as it is shown in Figure 24.

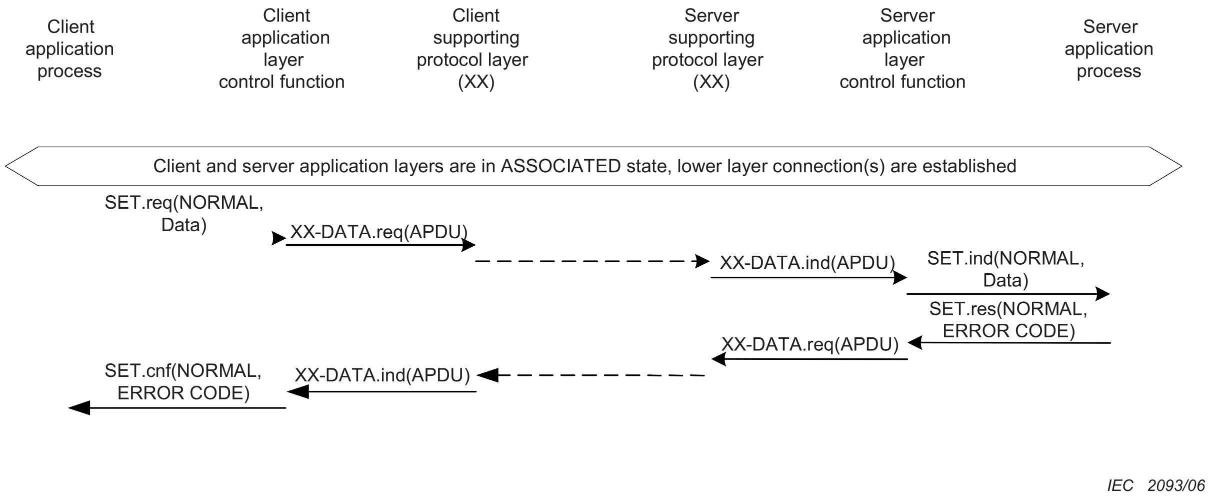

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s) are established
SET.req(NORMAL, Data)
XX-DATA.req(APDU)
XX-DATA.ind(APDU)
SET.ind(NORMAL, Data)
SET.res(NORMAL, ERROR CODE)
XX-DATA.req(APDU)
SET.cnf(NORMAL, ERROR CODE)
XX-DATA.ind(APDU)
IEC 2093/06

Figure 24 – MSC for the SET service in case of failure

NOTE The message sequence in the above figures applies only if the transferred data does not exceed the supported maximum size of the APDU. In order to be able to transfer longer data with the SET service, COSEM provides an application layer level protocol. This is described in 7.4.1.8.3.

The most complex behaviour is associated with the ACTION service, used for remote invocation of a method of a COSEM interface object in the server. The reason for this complexity is that the invocation of this method may imply data exchange in both client to server and server to client directions, and these data may be too long to fit into one APDU.

Figure 25 illustrates the message sequence chart in the case, when the required service can be granted by the server and the method invocation does not return data.

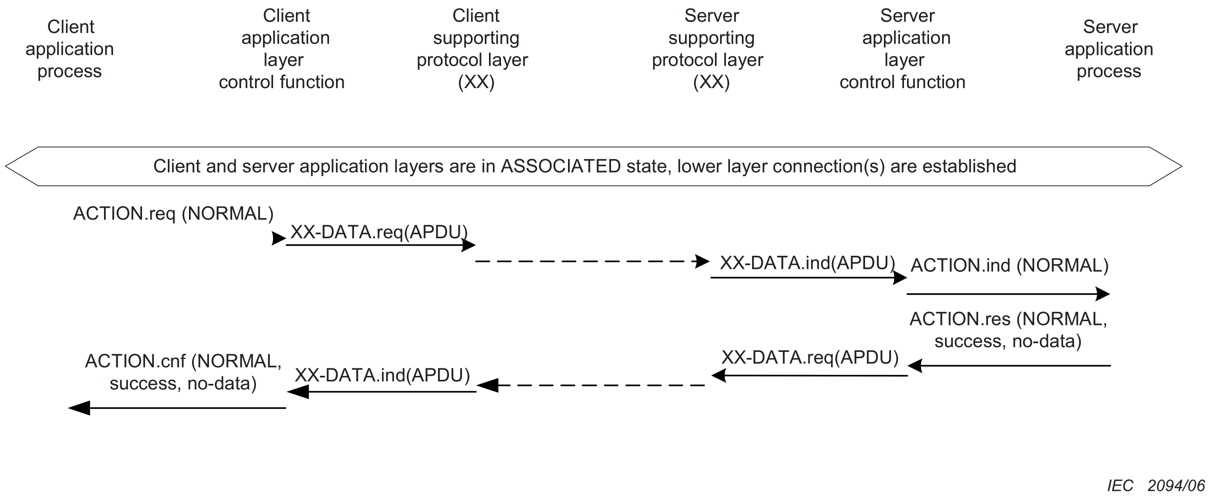

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s) are established
ACTION.req (NORMAL)
XX-DATA.req(APDU)
XX-DATA.ind(APDU)
ACTION.ind (NORMAL)
ACTION.res (NORMAL, success, no-data)
XX-DATA.req(APDU)
ACTION.cnf (NORMAL, success, no-data)
XX-DATA.ind(APDU)
IEC 2094/06

Figure 25 – MSC for the ACTION service (simplest case)

NOTE When either the parameters of the ACTION.request or the ACTION.response service do not fit in one APDU, the protocol defined in 7.4.1.8.4 for transferring long service parameters can be used.

7.4.1.2 Protocol for unconfirmed services

All client/server services may also be invoked in an unconfirmed manner within an established confirmed or unconfirmed AA. The following service primitives are supported:

- GET (.request/.indication);

- SET (.request/.indication);
- ACTION (.request/.indication).

The COSEM client may only invoke these.request primitives when an AA has already been established.

Three different kinds of destination addresses are possible: individual, group or broadcast. Depending on the destination address type, the receiving station shall handle incoming messages differently, as follows:

- XX-PDUs with an individual address of a COSEM logical device. If they are received within an established AA they shall be sent to the addressed COSEM logical device, otherwise shall be discarded;
- XX-PDUs with a group address of a group of COSEM logical devices. These shall be sent to the addressed group of COSEM logical devices. However, the received message shall be discarded if there is no association established between a client and the addressed group of COSEM logical devices;
- XX-PDUs with the broadcast address shall be sent to all addressed COSEM logical devices. However, the received message shall be discarded if there is no association 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 (e.g. broadcast address).

7.4.1.3 Protocol for the EventNotification service

This subclause specifies the protocol for the EventNotification.request service of the server application layer, specified in 6.6.3.2.7.

Events in metering equipment, like passing thresholds, fraud detection, or simply a counter overflow generally occur asynchronously to any operation. Depending on the implemented behaviour, the server may want to notify the client immediately.

As in the client/server environment the server is allowed to send information only upon a request from the client, the client may or may not gain knowledge about these events using COSEM client/server type services.

In order to ensure that the server can inform the client about events, a special, non-client/server type service, the EventNotification[19] service is available. As the EventNotification service is not a client-server type service, it may be sent out even when an AA is not established.

Upon invocation of the EventNotification.request service, the COSEM server application layer shall build an EventNotification.request APDU.

The possibilities to send out this APDU depend on the communication profile and the connection status of the lower layers. Therefore, the protocol of the EventNotification service is further discussed in Clause B.2 and Clause B.3.

In any case, in order to notify the client about the detection of an event by the server:

- the server shall use the EventNotification.request service invocation;
- this service invocation shall make the server application layer to build an Event Notification.request APDU;

[19] When short name referencing is used, the service is called InformationReport at the server side.

- this APDU shall be 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 communication profile used;.
- upon the reception of the EventNotification APDU, the client application layer shall generate an EventNotification.indication[20] service to the COSEM client application;
- by default, event notifications are sent from the management logical device (server) to the management AP (client).

7.4.1.4 Identifying a service invocation: using *Invoke_Id*

A complete confirmed data communication service sequence consists of the exchange of a.request and a.response type message (indicated to the peer protocol layer via the.indication and.confirmation service primitives). In the client/server model, requests are sent by the client and responses are sent by the server. As the client is allowed to send several.requests before receiving the.response for the previous one(s), it is necessary to make a reference in the.response message to the corresponding.request message. Otherwise, it is not possible to identify, which.request corresponds to a.response.

The Invoke_Id parameter identifies a.request and the corresponding.response. The value of this parameter is assigned by the client so that each.request primitive issued carries a different Invoke_Id. The server shall copy the Invoke_Id of the received.request message into the corresponding.response message.

The Invoke_Id is not present in the COSEM-OPEN services: these services are identified by their address parameters.

The EventNotification service – as it is not a client/server type service – does not contain Invoke_Id parameter either. There is no corresponding.response service, thus there is no need to use Invoke_Id. See also 8.3.

7.4.1.5 Using priority

COSEM defines two priority levels, 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 shall serve incoming service.requests in the order of their reception (FIFS, First In, First Served[21]). However, it is possible to request to be served first by setting the priority parameter of a.request to HIGH: a.request with priority HIGH shall be served before the previous requests with priority NORMAL. The.response primitive shall carry the same priority flag as that of the corresponding.request. Managing priority is a negotiable feature: its support is indicated by BIT 9 of the xDLMS conformance block.

NOTE If the feature is not supported, requests with HIGH priority shall be served with NORMAL priority.

7.4.1.6 Selective access

GET/SET services typically reference the entire attribute of a COSEM interface object. However for certain attributes, selective access to just a part of the attribute may be provided. The part of the attribute is identified by specific selective access parameters.[22] These

[20] At the client side, it is always EventNotification.indication, independently of the referencing scheme (logical name or short name) used at the server side.

[21] As service invocations are identified with an Invoke_Id, services with the same priority can be served in any order.

[22] Although the specification of these selection parameters is independent of the referencing method used (LN or SN), the use of these parameters is different for services using logical name (LN) referencing (GET/SET), and services using short name (SN) referencing (read/write). In this subclause selective access for the case of LN referencing is discussed. Selective access with SN referencing is called ‘parameterized access’, and is discussed in 7.4.2.7.

selective access parameters are defined as part of the attribute specification of the given COSEM interface class specification, see IEC 62056-62.

The selective access specification always starts with an access selector, followed by an access-specific access parameter list. In order to encode the selective access parameters, a 'selective-access-descriptor' type has been specified:

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

Using this type, the required parameters for selective access are included in the corresponding LN APDUs as an OPTIONAL field:

```
access-selection                 Selective-Access-Descriptor OPTIONAL
```

Selective access is a negotiable feature: its support is indicated by BIT 21 of the xDLMS conformance block.

7.4.1.7 Multiple references in the same service request

7.4.1.7.1 The Attribute_0 reference

GET/SET services typically reference one attribute of a COSEM interface object. The attribute referenced is identified by the value of the COSEM_Object_Attribute_Id parameter.

By convention, attributes are numbered from 1 to n, where Attribute_1 is the logical name of the COSEM interface object. Manufacturers may add proprietary methods and/or attributes to any object, using negative numbers. See also 4.1. of IEC 62056-62.

The value of 0 (zero) for the COSEM_Object_Attribute_Id (Attribute_0)[23] has a special meaning: it references all attributes with positive index (public attributes).

A GET.request service with COSEM_Object_Attribute_Id = 0 requests the value of all public attributes of the referenced object. The response to this request shall be a structure containing the value for all public attributes (data) in the order of their appearance in the given object specification. For attributes to which no access right is granted within the given association, or which cannot be accessed for any other reason, a null_data type NULL value shall be returned.

A SET.request service with COSEM_Object_Attribute_Id = 0 requests to set the value of all public attributes of the referenced object. The data sent with this request shall be a structure, containing for each public attribute, in the order of their appearance in the given object specification, either a value or a null_data type NULL value. The meaning of this NULL value is that the given attribute need not be set.

The response to this request shall be a structure containing the result for each public attribute (data-access-result) in the order of their appearance in the given object specification, indicating the success or failure of the requested SET operation. The response shall be carried by a SET-Response-With-List – type APDU.

Attribute_0 referencing is a negotiable feature: its support for the GET service is indicated by BIT 10, and for the SET service by BIT 8 of the xDLMS conformance block.

[23] The Attribute_0 feature cannot be applied when short name referencing is used.

7.4.1.7.2 Attribute reference list

A complete (LN) reference for an attribute includes the following parameters:

class-id	Cosem-Class-Id,
instance-id	Cosem-Object-Instance-Id,
attribute-id	Cosem-Object-Attribute-Id,
access-selection	Selective-Access-Descriptor **OPTIONAL**

A.request service may contain one such reference or a list of such references. Specification of the APDUs for the different types of.requests is given in 8.6.

7.4.1.8 Transferring long service parameters

7.4.1.8.1 Non-transparent and transparent transfer mechanisms

The service parameters of data communication services are transported by the APDUs, exchanged between the peer layers, in an encoded form. In some cases, the APDU can be longer than what the protocol is able to transmit in one piece. In order to be able to exchange such ‘long’ data, two transporting mechanisms are available:

a) long data transfer using an application level protocol. This mechanism can be used with any of the specified services (GET, SET and ACTION) and with any protocol profile and is specified in the following subclauses;

b) long data transfer in a transparent manner to the client application. This feature can be used only with lower layer protocols providing segmentation. As transparent long data transfer is specified only for the direction from the server to the client, the server side supporting protocol layer provides special services for this purpose to the server application layer. As these services are specific to the supporting protocol layer, handling these services is not within the scope of this standard – in other words, no specific application layer services and protocol are specified for this purpose. When the supporting protocol layer supports transparent long data transfer, the server side application layer implementation may be able to manage these services.

7.4.1.8.2 Application protocol for long data transfer with the GET service

Long data transfer with the GET service is specified only for the data in the GET.response service primitive.

The length of the encoded form of service parameters for selective access and/or multiple attribute references in the GET.request service shall not exceed the maximum allowed size of APDUs.

GET.request services shall be of type NORMAL or WITH-LIST. Upon reception of a GET.request, the server AP shall assemble the requested data. If the data fit into one APDU, the server AP shall invoke the GET.response service with NORMAL or WITH-LIST type, with the value(s) of the required attribute(s) as the result parameter.

If the data do not fit into one APDU and block transfer is supported (bit11 of the xDLMS conformance block), the server AP shall send the data in blocks.

First, the data shall be encoded, as if they would fit into one APDU. The result is a series of bytes, D_1,D_2,D_3,....D_N. The server shall then assemble a DataBlock-G data structure (see 8.3) with the following contents:

last-block (BOOLEAN)	= FALSE
block-number (Unsigned32)	= 0001
result (IMPLICIT OCTETSTRING)	= the first K bytes of the encoded data (D_1,D_2,D_3,....D_K)

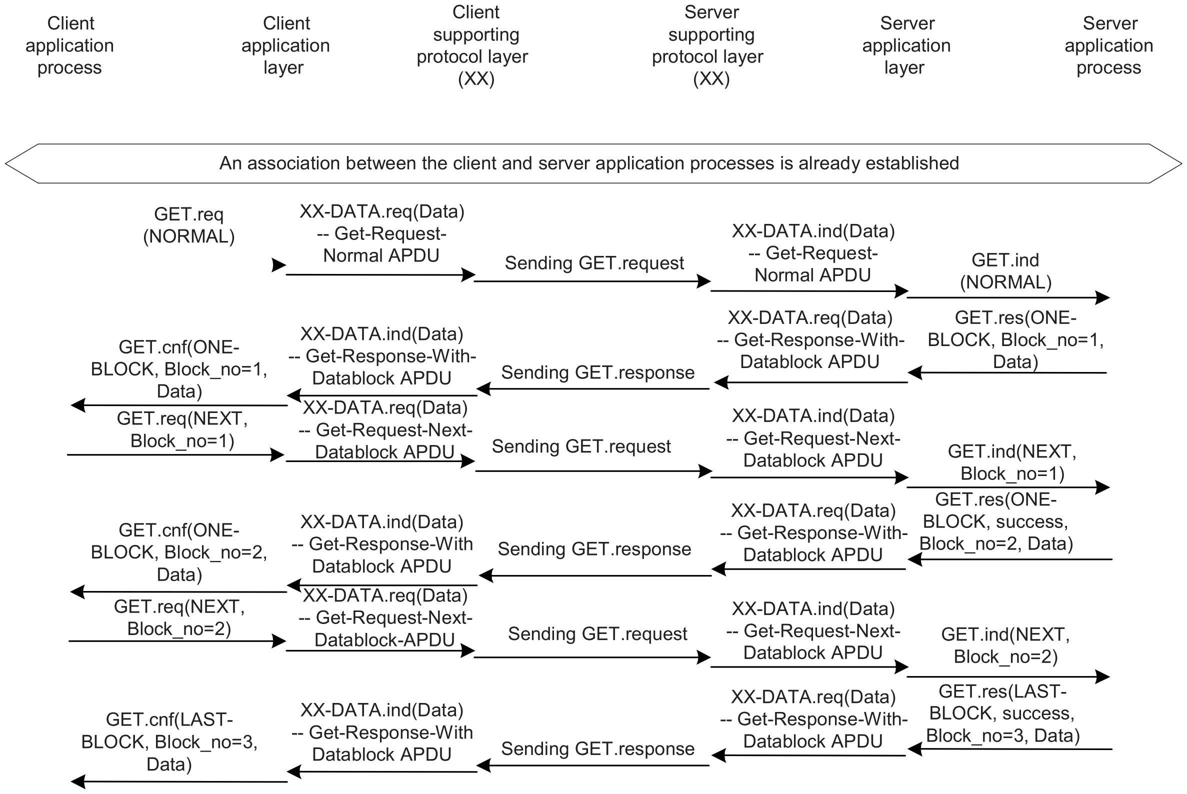
This DataBlock-G shall be the first part of the response. The server AP shall invoke the GET.response service with Response_type = ONE-BLOCK, with the Invoke_Id and priority parameters copied from the GET.request invocation received and with the DataBlock-G as result parameter.

Upon reception of this GET.response (signalled as.confirm), the client AP is informed that the response for its request does not fit into one APDU and shall proceed for the long data transfer. It shall store the data contents of the received APDU – (D_1,D_2,D_3,....D_K) – and shall acknowledge the received block by invoking the GET.request service with Request_type = NEXT and with the following parameters:

invoke-id-and-priority	=	the same as that for the first GET.request;
block-number	=	the same as the Block-number of the received data block.

When the server receives the acknowledgement, it shall prepare and send the next data block, including D_{K+1},D_{K+2},D_{K+3},....D_L, with block-number = 0002. This exchange of data blocks and acknowledgements shall normally continue until the last Data Block, including D_M,D_{M+1},D_{M+2},....D_N is sent. The last-block (BOOLEAN) parameter of this DataBlock-G sequence shall be set to TRUE and this data block shall not be acknowledged by the client. After the reception of the last data block, the long data transfer with the GET service is completed.

Figure 26 shows an example for the case, when the requested data can be sent in three parts, and the transfer is not aborted.


Client application process
Client application layer
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer
Server application process
An association between the client and server application processes is already established
GET.req (NORMAL)
XX-DATA.req(Data) -- Get-Request-Normal APDU
Sending GET.request
XX-DATA.ind(Data) -- Get-Request-Normal APDU
GET.ind (NORMAL)
GET.res(ONE-BLOCK, Block_no=1, Data)
XX-DATA.req(Data) -- Get-Response-With-Datablock APDU
Sending GET.response
XX-DATA.ind(Data) -- Get-Response-With-Datablock APDU
GET.cnf(ONE-BLOCK, Block_no=1, Data)
GET.req(NEXT, Block_no=1)
XX-DATA.req(Data) -- Get-Request-Next-Datablock APDU
Sending GET.request
XX-DATA.ind(Data) -- Get-Request-Next-Datablock APDU
GET.ind(NEXT, Block_no=1)
GET.res(ONE-BLOCK, success, Block_no=2, Data)
XX-DATA.req(Data) -- Get-Response-With-Datablock APDU
Sending GET.response
XX-DATA.ind(Data) -- Get-Response-With Datablock APDU
GET.cnf(ONE-BLOCK, Block_no=2, Data)
GET.req(NEXT, Block_no=2)
XX-DATA.req(Data) -- Get-Request-Next-Datablock-APDU
Sending GET.request
XX-DATA.ind(Data) -- Get-Request-Next-Datablock APDU
GET.ind(NEXT, Block_no=2)
GET.res(LAST-BLOCK, success, Block_no=3, Data)
XX-DATA.req(Data) -- Get-Response-With-Datablock APDU
Sending GET.response
XX-DATA.ind(Data) -- Get-Response-With Datablock APDU
GET.cnf(LAST-BLOCK, Block_no=3, Data)

IEC 2095/06

Figure 26 – Long data with the GET service in three data blocks

The server may generate the complete response (D_1,D_2,D_3,....D_N) upon the receipt of the first GET.request, or it could generate the series of data blocks of the response dynamically (on the fly).

If any error occurs during the long data transfer, the transfer shall be aborted. Error cases are as follows:

- the server is not able to provide the next block of data for any reason. In this case, it shall send back a Get-Response-With-Datablock APDU. In the DataBlock-G, the last-block shall be set to TRUE, the block-number shall be equal to the block_number confirmed in the previous Get-Request-Next APDU sent by the client + 1 and the result shall contain a data-access-result indicating the reason for the failure;
- the Block_Number parameter in a GET.indication of type NEXT is not equal to the Block_Number parameter of the last block sent by the server. The server shall interpret this case, as if the client would like to abort the ongoing transfer. The server, instead of sending back the next data block, shall send a Get-Response-With-Datablock APDU. In the DataBlock-G, the last-block shall be set to TRUE, the block-number shall be equal to the block-number received in the GET.request with type NEXT and the result shall be data-access-result = long-get-aborted;
- the server may receive a GET.indication of type NEXT when no long data transfer is in progress. In this case, the response shall be a Get-Response-With-Datablock APDU. In the DataBlock-G, the last-block shall be set to TRUE, the block-number shall be equal to the block-number received in the GET.request of type NEXT and the result shall be data-access-result = no-long-get-in-progress.

During the data exchange, the Invoke-Id-and-Priority parameter shall be the same for all APDUs. If during a long data transfer another service request is received, it shall be served according to the priority rules.

If for any reason the server is not able to send back a GET.response of type LAST-BLOCK, it shall then send a GET.response of type NORMAL, with a data-access-result indicating the reason for the failure.

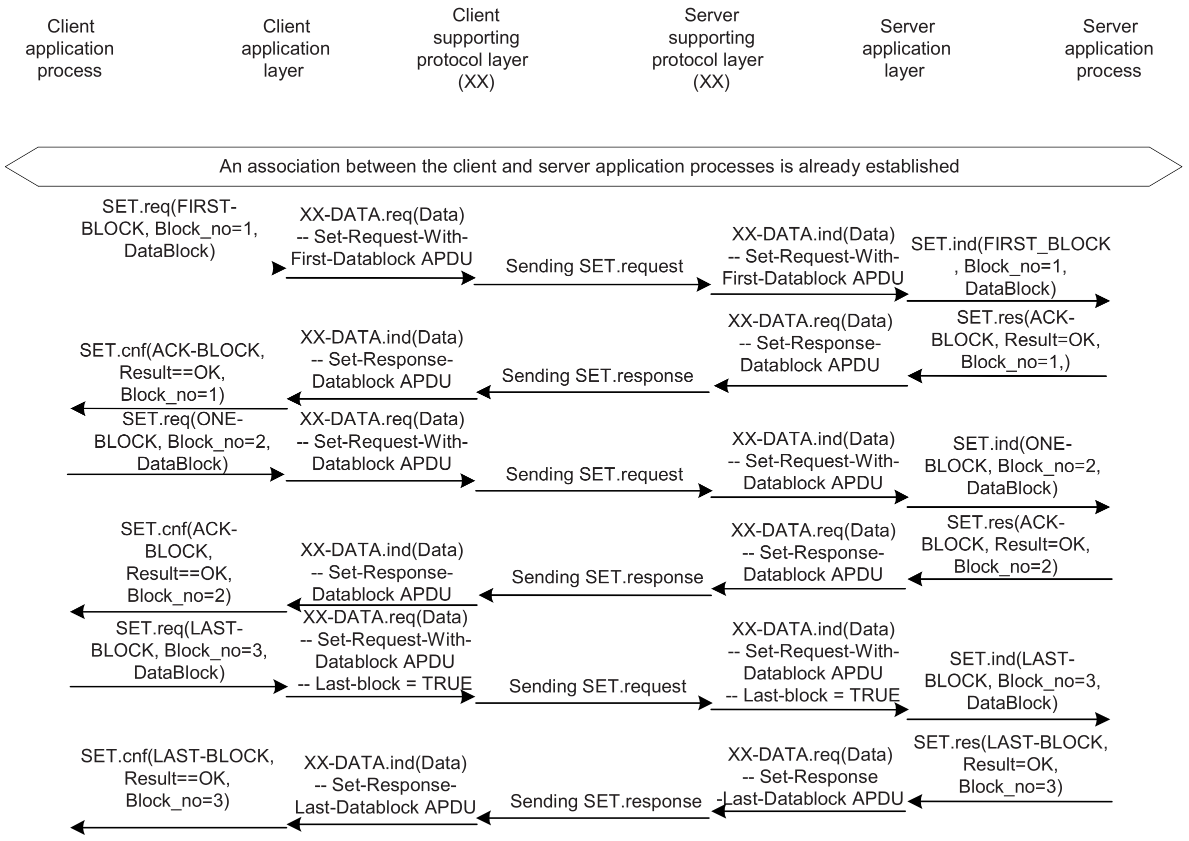
Block transfer with the GET service is a negotiable feature: its support is indicated by BIT 11 of the xDLMS conformance block.

7.4.1.8.3 Application protocol for long data transfer with the SET service

Long data transfer with the SET service is specified only for the data in the SET.request service primitive.

The length of the encoded form of service parameters for selective access and/or multiple attribute references in the SET.response service shall not exceed the maximum allowed APDU size.

The main difference between the GET and the SET.request services is that the client, before issuing the first SET.request service invocation, already knows whether a long data transfer is required or not. If long data transfer is required – and if block transfer is supported (bit12 of the xDLMS conformance block) – the first SET.request service shall already contain the first data block.


Client application process
Client application layer
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer
Server application process
An association between the client and server application processes is already established
SET.req(FIRST-BLOCK, Block_no=1, DataBlock)
XX-DATA.req(Data) -- Set-Request-With-First-Datablock APDU
Sending SET.request
XX-DATA.ind(Data) -- Set-Request-With-First-Datablock APDU
SET.ind(FIRST_BLOCK , Block_no=1, DataBlock)
SET.res(ACK-BLOCK, Result=OK, Block_no=1,)
XX-DATA.req(Data) -- Set-Response-Datablock APDU
Sending SET.response
XX-DATA.ind(Data) -- Set-Response-Datablock APDU
SET.cnf(ACK-BLOCK, Result==OK, Block_no=1)
SET.req(ONE-BLOCK, Block_no=2, DataBlock)
XX-DATA.req(Data) -- Set-Request-With-Datablock APDU
Sending SET.request
XX-DATA.ind(Data) -- Set-Request-With-Datablock APDU
SET.ind(ONE-BLOCK, Block_no=2, DataBlock)
SET.res(ACK-BLOCK, Result=OK, Block_no=2)
XX-DATA.req(Data) -- Set-Response-Datablock APDU
Sending SET.response
XX-DATA.ind(Data) -- Set-Response-Datablock APDU
SET.cnf(ACK-BLOCK, Result==OK, Block_no=2)
SET.req(LAST-BLOCK, Block_no=3, DataBlock)
XX-DATA.req(Data) -- Set-Request-With-Datablock APDU -- Last-block = TRUE
Sending SET.request
XX-DATA.ind(Data) -- Set-Request-With-Datablock APDU -- Last-block = TRUE
SET.ind(LAST-BLOCK, Block_no=3, DataBlock)
SET.res(LAST-BLOCK, Result=OK, Block_no=3)
XX-DATA.req(Data) -- Set-Response -Last-Datablock APDU
Sending SET.response
XX-DATA.ind(Data) -- Set-Response-Last-Datablock APDU
SET.cnf(LAST-BLOCK, Result==OK, Block_no=3)

IEC 2096/06

Figure 27 – Long data transfer in three data blocks with the SET service

In both cases, the first SET.request service invocation may contain a single attribute reference, or a list of attribute references. Although the data contents of the SET.request may be transmitted in several data blocks, attribute reference(s) shall be present only in the first SET.request invocation service.

The client assembles data blocks in the same way as described in the previous clause. Data blocks are placed into DataBlock-SA structures as raw-data and are sent to the server.

Each data block shall be acknowledged by the server with a SET.response service primitive, of Response_type = ACK_BLOCK. The Result parameter indicates only the good (or not good) reception of the data block.

The server shall acknowledge the whole SET.request service invocation after the reception of the last data block, with a SET.response service primitive of type LAST-BLOCK or LAST-BLOCK-WITH-LIST. The result parameter in this service indicates the result of the SET operation.

If any error occurs during the long data transfer, the transfer shall be aborted. Error cases are as follows:

- the server is not able to handle the received next DataBlock-SA, for any reason. In this case, it shall send back a Set-Response-Last-Datablock APDU, with a result parameter indicating the reason for aborting the transfer, and shall consider the transfer terminated;

- the Block_Number parameter in a received SET.indication of type ONE BLOCK is not equal to the Block_Number parameter expected by the server (last received + 1). The server shall interpret this as if the client would like to abort the ongoing transfer. The server shall send back a Set-Response-Last-Datablock APDU with Data-Access-Result = long-set-aborted;
- the server may receive a Set-Request-With-Datablock APDU when no long data transfer is in progress. In this case, the response shall be a Set-Response-Last-Datablock APDU with Data-Access-Result = no-long-set-in-progress.

During the data exchange, the Invoke-Id-and-Priority parameter shall be the same for all APDUs. If during a long data transfer another service request is received, it shall be served according to the priority rules.

If for any reason the server is not able to send back a SET.response of type LAST-BLOCK, it shall then send a SET.response of type NORMAL, with a data-access-result indicating the reason for the failure.

Block transfer with the SET service is a negotiable feature: its support is indicated by BIT 12 of the xDLMS conformance block.

7.4.1.8.4 Application protocol for long data transfer with the ACTION service

Remote invocation of a COSEM interface object method using the ACTION service may require parameters, which in their encoded form are longer than the maximum APDU size allowed. On the other hand, a method invocation may cause the server to send back data, which do not fit into one APDU either. Therefore, long data transfer with the ACTION service is available for both directions.

Long data transfer in the two directions shall take place in two stages:

- first, the client shall transmit the whole ACTION.request to the server (in parameter blocks, if it is required);
- second, the server shall transmit the whole ACTION.response to the client (in parameter blocks, if it is required).

Similarly to the SET service, the client, before issuing the first ACTION.request service invocation, already knows whether a long data transfer is required or not. If long data transfer is required – and if block transfer is supported (bit13 of the xDLMS conformance block) – the first ACTION.request service invocation shall already contain the first data block.

In both cases, the first ACTION.request invocation may contain a single method reference, or a list of method references. Although the data contents of the ACTION.request (the Method_Invocation_Parameters) may be transmitted in several data blocks, method reference(s) shall be present only in the first invocation of the ACTION.request service.

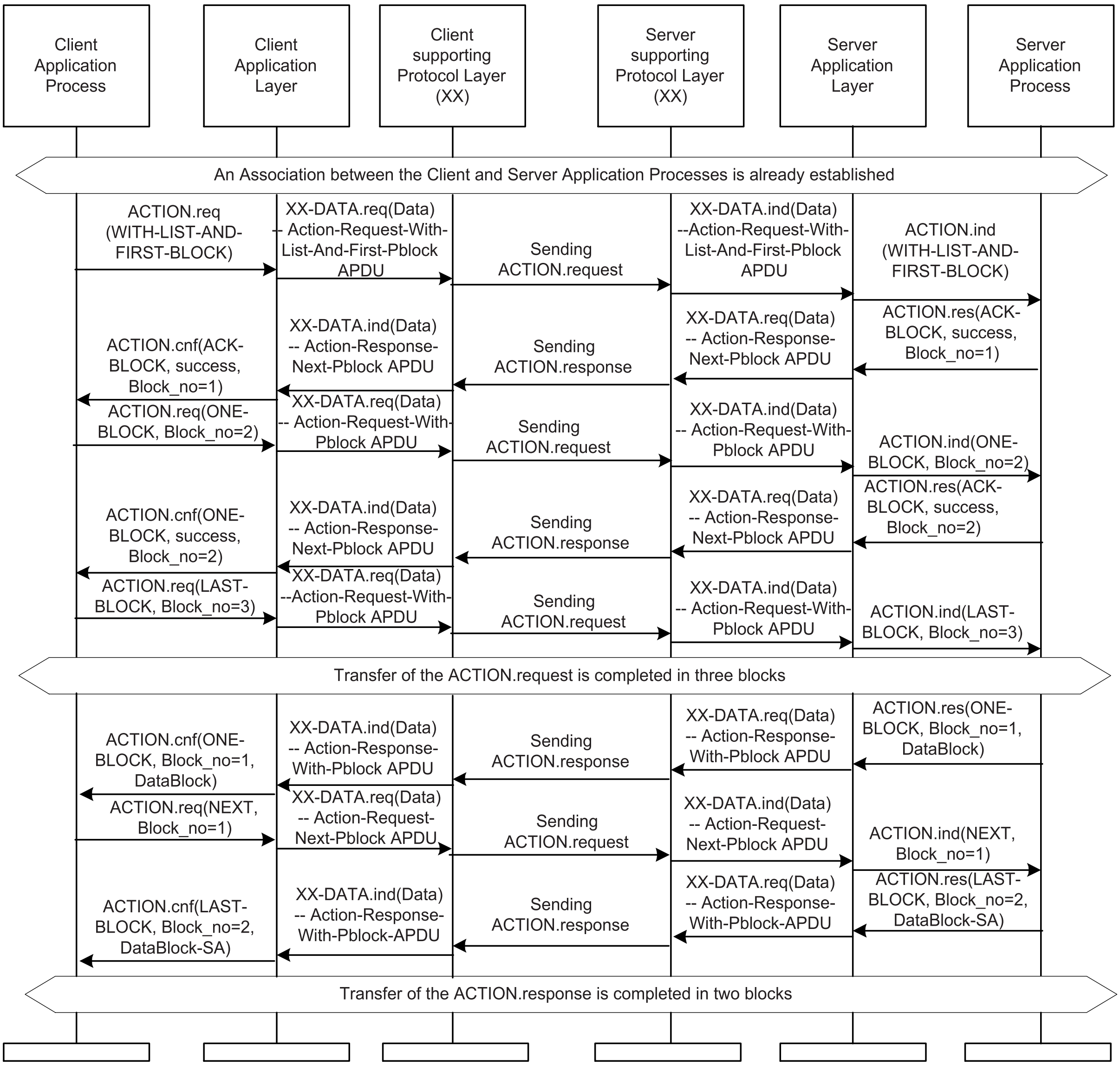
The client assembles the data block in the same way as it is described in 7.4.1.8.3. Data blocks shall be placed into DataBlock-SA data structures as raw-data, and sent to the server.

Once the complete ACTION.request is transmitted and the server has locally activated all required methods, the server shall invoke the ACTION.response service. The response to one method invocation shall contain a SEQUENCE of two parameters: the first parameter indicates the result of the method invocation (result), and the second, optional one carries the data required by the ACTION invocation. (see 8.3).

The ACTION.response service primitive may take one of the following forms:

- NORMAL: the corresponding ACTION.request contained only one method reference, and the response fits into one APDU;
- WITH-LIST: the corresponding ACTION.request contained a list of method references, and the complete response fits into one APDU;
- BLOCK: the corresponding ACTION.request could contain only one or a list of method references (it determines only the contents of the parameter block). The response to that ACTION.request does not fit into one APDU. In this case, the server shall initiate a long data transfer, which shall take place similarly as it is described for the GET service in 7.4.1.8.2.

Figure 28 illustrates a case, when the client sends an ACTION.request, including multiple method references in three blocks, and the server sends the response in two blocks. Similarly to the GET service, the service is completed when the last block of the response is sent by the server. This block is not acknowledged by the client.


Client Application Process
Client Application Layer
Client supporting Protocol Layer (XX)
Server supporting Protocol Layer (XX)
Server Application Layer
Server Application Process
An Association between the Client and Server Application Processes is already established
ACTION.req (WITH-LIST-AND-FIRST-BLOCK)
XX-DATA.req(Data) -- Action-Request-With-List-And-First-Pblock APDU
Sending ACTION.request
XX-DATA.ind(Data) --Action-Request-With-List-And-First-Pblock APDU
ACTION.ind (WITH-LIST-AND-FIRST-BLOCK)
ACTION.res(ACK-BLOCK, success, Block_no=1)
XX-DATA.req(Data) -- Action-Response-Next-Pblock APDU
Sending ACTION.response
XX-DATA.ind(Data) -- Action-Response-Next-Pblock APDU
ACTION.cnf(ACK-BLOCK, success, Block_no=1)
ACTION.req(ONE-BLOCK, Block_no=2)
XX-DATA.req(Data) -- Action-Request-With-Pblock APDU
Sending ACTION.request
XX-DATA.ind(Data) -- Action-Request-With-Pblock APDU
ACTION.ind(ONE-BLOCK, Block_no=2)
ACTION.res(ACK-BLOCK, success, Block_no=2)
XX-DATA.req(Data) -- Action-Response-Next-Pblock APDU
Sending ACTION.response
XX-DATA.ind(Data) -- Action-Response-Next-Pblock APDU
ACTION.cnf(ONE-BLOCK, success, Block_no=2)
ACTION.req(LAST-BLOCK, Block_no=3)
XX-DATA.req(Data) --Action-Request-With-Pblock APDU
Sending ACTION.request
XX-DATA.ind(Data) -- Action-Request-With-Pblock APDU
ACTION.ind(LAST-BLOCK, Block_no=3)
Transfer of the ACTION.request is completed in three blocks
ACTION.res(ONE-BLOCK, Block_no=1, DataBlock)
XX-DATA.req(Data) -- Action-Response-With-Pblock APDU
Sending ACTION.response
XX-DATA.ind(Data) -- Action-Response-With-Pblock APDU
ACTION.cnf(ONE-BLOCK, Block_no=1, DataBlock)
ACTION.req(NEXT, Block_no=1)
XX-DATA.req(Data) -- Action-Request-Next-Pblock APDU
Sending ACTION.request
XX-DATA.ind(Data) -- Action-Request-Next-Pblock APDU
ACTION.ind(NEXT, Block_no=1)
ACTION.res(LAST-BLOCK, Block_no=2, DataBlock-SA)
XX-DATA.req(Data) -- Action-Response-With-Pblock-APDU
Sending ACTION.response
XX-DATA.ind(Data) -- Action-Response-With-Pblock-APDU
ACTION.cnf(LAST-BLOCK, Block_no=2, DataBlock-SA)
Transfer of the ACTION.response is completed in two blocks

IEC 2097/06

Figure 28 – Long data transfer with the ACTION service

The first part of the long transfer (client->server) is similar to the SET service and the second part of the transfer (server->client) is similar to the GET service: the ACTION service can be considered as a combined SET/GET service.

If any error occurs during the long data transfer, the transfer shall be aborted. Error cases are the same as the cases described in 7.4.1.8.2 and 7.4.1.8.3.

During the data exchange, the Invoke-Id-and-Priority parameter shall be the same for all APDUs. If during a long data transfer another service request is received, it shall be served according to the priority rules.

If for any reason the server is not able to send back an ACTION.response of type LAST-BLOCK, it shall then send an ACTION.response of type NORMAL, with a data-access-result indicating the reason for the failure.

Block transfer with the ACTION service is a negotiable feature: its support is indicated by BIT 13 of the xDLMS conformance block. If block transfer is supported, it should be supported in both directions.

7.4.2 Protocol for the xDLMS services using SN referencing

7.4.2.1 Protocol for confirmed services

The following services are supported when invoked in a confirmed manner:

- Read;
- Write.

As it is defined in 5.3.2, the COSEM client AP always invokes data communication services with logical name references. When the server uses short name referencing, the client application layer shall transform service invocations using LN referencing to service invocations using SN referencing. This is done by the short name mapper service element of the ASO. The mapping is defined in 6.5.5.2. These SN referencing services shall then be transmitted to the server.

Upon the receipt of a service request, the server AP checks whether the service can be provided or not (validity, client access right, availability, etc.). If everything is OK, it locally applies the required service to the corresponding ‘real’ object. The COSEM server AP generates then the appropriate response message using SN referencing. This message shall be re-translated to the appropriate service using LN referencing by the client application layer.

A complete message sequence for the ReadRequest/Response service invocation is shown in Figure 29.

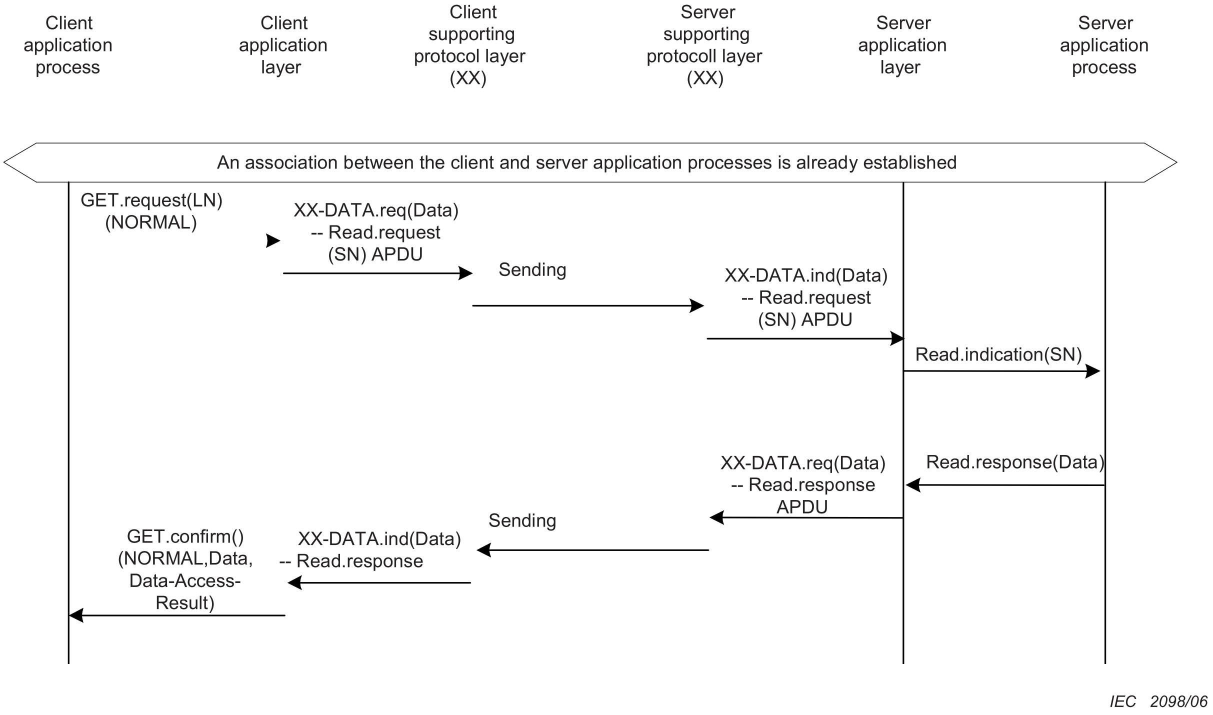

Client application process
Client application layer
Client supporting protocol layer (XX)
Server supporting protocoll layer (XX)
Server application layer
Server application process
An association between the client and server application processes is already established
GET.request(LN) (NORMAL)
XX-DATA.req(Data) -- Read.request (SN) APDU
Sending
XX-DATA.ind(Data) -- Read.request (SN) APDU
Read.indication(SN)
XX-DATA.req(Data) -- Read.response APDU
Read.response(Data)
Sending
GET.confirm() (NORMAL,Data, Data-Access-Result)
XX-DATA.ind(Data) -- Read.response
IEC 2098/06

Figure 29 – MSC for the ReadRequest/Response services

NOTE The message sequence applies only if the size of data to be transferred does not exceed the maximum APDU size supported. Non-transparent long-data transfer (see 7.4.1.8) is not available with short name referencing.

7.4.2.2 Protocol for unconfirmed services

For unconfirmed requests, the following service is available:

- UnconfirmedWriteRequest.

The COSEM client may only invoke this.request primitive when an AA has already been established.

Depending on the communication profile, the APDU corresponding to this.request may be transported using the connection-oriented or connectionless data services of the supporting protocol layer.

Three different kinds of destination addresses are possible: individual, group or broadcast. Depending on the destination address type, the receiving station shall handle incoming messages differently, as follows:

- XX-PDUs with an individual address of a COSEM logical device. If they are received within an established AA they shall be sent to the addressed COSEM logical device, otherwise they shall be discarded;
- XX-PDUs with a group address of a group of COSEM logical devices. These shall be sent to the addressed group of COSEM logical devices. However, the received message shall be discarded if there is no association established between a client and the addressed group of COSEM logical devices;
- XX-PDUs with the broadcast address shall be sent to all addressed COSEM logical devices. However, the received message shall be discarded if there is no association established between a client and the All-station address;

- XX-PDUs with the broadcast address shall be sent to all addressed COSEM logical devices. However, the received message shall be discarded if there is no association 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 (e.g. broadcast address).

7.4.2.3 Protocol for the InformationReport service

This subclause specifies the protocol for the EventNotification.request service of the server application layer, specified in 6.6.3.3.6.

As, unlike the EventNotification.request service, the InformationReportRequest service does not contain the optional Application_Addresses parameter, the information report is always sent by the Server Management Logical Device to the Client Management AP.

Upon invocation of the InformationReport.request service, the COSEM server application shall build an InformationReportRequest APDU. This APDU shall be 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 Information ReportRequest service is further discussed in Clause B.2 and Clause B.3.

Upon the receipt of an InformationReport APDU, the client application layer shall generate an EventNotification.indication primitive to the client AP.

7.4.2.4 Mapping of an InformationReport service to an EventNotification.indication service

The InformationReport service description and the description of the corresponding APDU can be found in IEC 61334-4-41 as:

```
InformationReportRequest::= SEQUENCE{
        current-time                        GeneralizedTime OPTIONAL,
        variable-access-specification       SEQUENCE OF VariableAccessSpecification,
        list-of-data
};
```

where the current-time parameter defines the time instance when the event occurred. The variable-access-specification parameter contains a list of short names describing the attributes, which contain information relevant to the event. The list-of-data parameter carries the values of the attributes defined in the variable-access-specification.

While the InformationReportRequest APDU may carry several attribute names and their contents, the EventNotification.ind (see 6.5.4.1.) contains only one attribute reference. Therefore, in the case when the InformationReportRequest APDU contains more than one attribute, it must be mapped to several EventNotification.ind services. The service parameters are mapped as follows:

Table 10 – Mapping between the EventNotification and InformationReport services

EventNotification.ind	InformationReport
Time (optional)	current-time (optional)
COSEM_Class_Id, COSEM_Object_Instance_Id, COSEM_Object_Attribute_Id	variable–name (as part of the variable-access-specification)
attribute value	Data (as part of list-of-data)

7.4.2.5 Identifying a service invocation

This feature is not provided in conjunction with SN referencing.

7.4.2.6 Using priority

A priority feature is not provided in conjunction with SN referencing. The server treats the services on a “first come first served” basis.

7.4.2.7 Selective access

READ/WRITE services typically reference the entire attribute of a COSEM interface object. However, for certain attributes selective access to just part of the attribute may be provided. The part of the attribute is identified by specific selective access parameters. These selective access parameters are defined as part of the attribute specification of the COSEM interface object specification. See in IEC 62056-62.

The selective access specification starts with an optional access selector, followed by an access-specific access parameter list. In order to encode the selective access parameters into the Read/WriteRequest service, the “VariableAccessSpecification” part of the DLMS specification has been extended as specified in 6.6.3.3.1.

Parameterized access is a negotiable feature. Its support is indicated by BIT 18 of the xDLMS conformance block.

7.4.2.8 Multiple references in the same service invocation

Reference to multiple short names is possible with the Read, Write and UnconfirmedWrite Services (see in IEC 61334-4-41).

Support of multiple references is a negotiable feature. It is indicated by the BIT 14 of the xDLMS conformance block.

7.4.2.9 Transferring long service parameters

The availability of transparent transfer of long data depends on the communication profile. See B.2.6.5 and B.3.6.5.

8 Specification of COSEM data types

8.1 The COSEM APDUs

In addition to the APDUs defined in IEC 61334-4-41, some new APDUs have been specified for COSEM in a manner that they are not in conflict with the DLMS PDUs. Thus, the APDUs used in COSEM are the following:

```
COSEMpdu     ::= CHOICE {

  -- standardized DLMS PDUs used in COSEM
  -- DLMS PDUs (no encryption selected[24])
      initiateRequest              [1] IMPLICIT         InitiateRequest,
      readRequest                  [5] IMPLICIT         ReadRequest,
      writeRequest                 [6] IMPLICIT         WriteRequest,

      initiateResponse             [8] IMPLICIT         InitiateResponse,
```

[24] Ciphered application contexts will use the corresponding ciphered DLMS PDUs.

```
      readResponse                  [12] IMPLICIT        ReadResponse,
      writeResponse                 [13] IMPLICIT        WriteResponse,

      confirmedServiceError         [14]                 ConfirmedServiceError,

      unconfirmedWriteRequest       [22] IMPLICIT        UnconfirmedWriteRequest,
      informationReportRequest      [24] IMPLICIT        InformationReportRequest,

-- the four ACSE APDUs
      aarq      AARQ-apdu
      aare      AARE-apdu
      rlrq      RLRQ-apdu           -- OPTIONAL
      rlre      RLRE-apdu           -- OPTIONAL

-- APDUs used for data communication services using LN referencing
      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,

-- global ciphered pdus
      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,

-- dedicated ciphered pdus
      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 respsonse pdu
      exception-response            [216] IMPLICIT OCTET STRING
```

8.2 The ACSE 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] IMPLICIT Association-information OPTIONAL
  }

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] IMPLICIT Association-information OPTIONAL
  }

RLRQ-apdu    ::= [APPLICATION 2] IMPLICIT SEQUENCE
                                                 -- [APPLICATION 2] == [ 62H ] = [ 98 ]
        {
              reason                     [0]      IMPLICIT Release-request-reason OPTIONAL,
              user-information           [30]     IMPLICIT Association-information OPTIONAL
        }

RLRE-apdu    ::= [APPLICATION 3] IMPLICIT SEQUENCE
                                                 -- [APPLICATION 3] == [ 63H ] = [ 99 ]
        {
              reason                     [0]      IMPLICIT Release-response-reason OPTIONAL,
              user-information           [30]     IMPLICIT Association-information OPTIONAL
}
ACSE-requirements             ::= BIT STRING { authentication (0) }

Application-context-name      ::= OBJECT IDENTIFIER

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        ::= OCTETSTRING[25]

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)
}
```

[25] In ISO/IEC 8650-1 the association-information field is specified as::= SEQUENCE OF EXTERNAL. For COSEM, this field shall always contain the A-XDR encoded DLMS-Initiate.request /.response pdus, (or a ConfirmedServiceError-pdu when the requested xDLMS context is not supported by the server) as a BER encoded OCTETSTRING.

8.3 Useful types

```
-- Useful Types

Integer8        ::= INTEGER(-128…127)
Integer16       ::= INTEGER(-32 768…32 767)
Integer32       ::= INTEGER(-2 147 483 648…2 147 483 647)
Integer64       ::= INTEGER(-2^63… 2^63-1)
Unsigned8       ::= INTEGER(0…255)
Unsigned16      ::= INTEGER(0…65 535)
Unsigned32      ::= INTEGER(0…4 294 967 295)
Unsigned64      ::= INTEGER(0…2^64-1)

Invoke-Id-And-Priority          ::= BIT STRING (SIZE(8))
                                {
                                invoke_id       (0…3),
                                reserved        (4…5),
                                service_class   (6),      -- 0 = Unconfirmed, 1 = Confirmed
                                priority (7)              -- 0 = normal; 1 = high
}
```

NOTE In the 3-layer, connection-oriented, HDLC based profile bit 6 is not relevant, as the service_class information is conveyed by the HDLC fame type carrying the APDU.

```
ObjectName    ::= Integer16

Cosem-Class-Id                  ::= Unsigned16

Cosem-Object-Instance-Id        ::= OCTET STRING

Cosem-Object-Attribute-Id       ::= Integer8

Cosem-Object-Method-Id          ::= Integer8

Cosem-Date-Time                 ::=OCTET STRING (SIZE(12))[26]

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
  }

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

[26] The content of this OCTET STRING is an encoded as date_time, as specified in IEC 62056-62.

```
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
  }

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

ConfirmedServiceError          ::= CHOICE
  {
 -- tag 0 is reserved
        initiateError          [1]      ServiceError,
        getStatus              [2]      ServiceError,[27]
        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),
```

[27] Greyed lines are not applicable within the DLMS context.

```
            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
                                              --  (refer to companion specification)
            service-unsupported         (2)   --  as described in the conformance block
    },
definition              [4]                     IMPLICIT ENUMERATED {
-- object bound troubles in a service
other                           (0),
object-undefined                (1), -- object not defined at the VDE
object-class-inconsistent       (2), -- class of object incompatible with asked service
object-attribute-inconsistent   (3)  -- object attributes are inconsistent
    },
access                  [5]                     IMPLICIT ENUMERATED {
        -- object access error
            other                       (0),
            scope-of-access-violated    (1), -- access denied through authorization reason
            object-access-invalid       (2), -- access incompatible with object attribute
            hardware-fault              (3), -- access fail for hardware reason
            object-unavailable          (4)  -- VDE hands object for unavailable
            },

initiate                [6]                     IMPLICIT ENUMERATED {
            -- initiate service error
            other                       (0),
            dlms-version-too-low        (1), -- proposed DLMS version too low
            incompatible-conformance    (2), -- proposed services not sufficient
            pdu-size-too-short          (3), -- proposed pdu size too short
            refused-by-the-VDE-Handler  (4)  -- VAA creation impossible or not allowed
            },
load-data-set           [7]                     IMPLICIT ENUMERATED {
 -- data set load services error
            other                       (0),
            primitive-out-of-sequence   (1), -- according to the DataSet
                                             -- loading state transitions
            not-loadable                (2), -- loadable attribute set to FALSE
            dataset-size-too-large      (3), -- evaluated Data Set size too large
            not-awaited-segment         (4), -- proposed segment not awaited
            interpretation-failure      (5), -- segment interpretation error
            storage-failure             (6), -- segment storage error
            data-set-not-ready          (7) -- Data Set not in correct state for uploading
            },
-- change-scope         [8]                     IMPLICIT      reserved.
task                    [9]                     IMPLICIT ENUMERATED {
            -- TI services error
            other                       (0),
            no-remote-control           (1), -- Remote Control parameter set to FALSE
            ti-stopped                  (2), -- TI in stopped state
            ti-running                  (3), -- TI in running state
            ti-unusable                 (4) -- TI in unusable state
        }
        -- other                [10]                    IMPLICIT ENUMERATED
        }

Data    ::= CHOICE
  {
        null-data               [0]     IMPLICIT NULL,
        array                   [1]     IMPLICIT SEQUENCE OF Data,
        structure               [2]     IMPLICIT SEQUENCE OF Data,
        boolean                 [3]     IMPLICIT BOOLEAN,
        bit-string              [4]     IMPLICIT BIT STRING,
        double-long             [5]     IMPLICIT Integer32,
        double-long-unsigned    [6]     IMPLICIT Unsigned32,
        octet-string            [9]     IMPLICIT OCTET STRING,
        visible-string          [10]    IMPLICIT VisibleString,
        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 ENUMERATED,
        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))

        don't-care              [255]   IMPLICIT NULL
  }


TypeDescription         ::= CHOICE
  {
        null-data               [0]     IMPLICIT NULL,
        array                   [1]     IMPLICIT SEQUENCE {
                                                number-of-elements      Unsigned16,
                                                type-description        TypeDescription
        }
        structure               [2]     IMPLICIT SEQUENCE OF TypeDescription,
        boolean                 [3]     IMPLICIT NULL,
        bit-string              [4]     IMPLICIT NULL,
        double-long             [5]     IMPLICIT NULL,
        double-long-unsigned    [6]     IMPLICIT NULL,
        octet-string            [9]     IMPLICIT NULL,
        visible-string          [10]    IMPLICIT NULL,
        bcd                     [13]    IMPLICIT NULL,
        integer                 [15]    IMPLICIT NULL,
        long                    [16]    IMPLICIT NULL,
        unsigned                [17]    IMPLICIT NULL,
        long-unsigned           [18]    IMPLICIT NULL,
        long64                  [20]    IMPLICIT NULL,
        long64-unsigned         [21]    IMPLICIT NULL,
        enum                    [22]    IMPLICIT NULL,
        float32                 [23]    IMPLICIT NULL,
        float64                 [24]    IMPLICIT NULL,
        date_time               [25]    IMPLICIT NULL
        date                    [26]    IMPLICIT NULL
        time                    [27]    IMPLICIT NULLL
        don't-care              [255]   IMPLICIT NULL
  }
```

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

DataBlock-SA  ::= SEQUENCE        -- SA == DataBlock for the SET.request and
                                  --          ACTION.request/.response services
  {
        last-block              BOOLEAN,
        block-number            Unsigned32,
        raw-data                OCTETSTRING
  }

Data-Access-Result    ::= ENUMERATED
  {
        success                         (0),
        hardware-fault                  (1),
        temporary-failure               (2),
        read-write-denied               (3),
        object-undefined                (4),
        object-class-inconsistent       (9),
        object-unavailable              (11),
        type-unmatched                  (12),
        scope-of-access-violated        (13),
        data-block-unavailable          (14),
        long-get-aborted                (15),
        no-long-get-in-progress         (16),
        long-set-aborted                (17),
        no-long-set-in-progress         (18),
        other-reason                    (250)
  }

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

8.4 The xDLMS-Initiate.request/response/ConfirmedServiceError PDUs

```
xDLMS-Initiate.request          ::= SEQUENCE
  {
        dedicated-key                       OCTET STRING OPTIONAL,
     -- shall not be encoded in DLMS without encryption
        response-allowed                    BOOLEAN DEFAULT TRUE,
        proposed-quality-of-service         [0] IMPLICIT Integer8 OPTIONAL,
        proposed-dlms-version-number        Unsigned8,
        proposed-conformance                Conformance,
        client-max-receive-pdu-size         Unsigned16
  }

xDLMS-Initiate.response         ::= SEQUENCE
  {
        negotiated-quality-of-service       [0] IMPLICIT Integer8 OPTIONAL,
        negotiated-dlms-version-number      Unsigned8,
        negotiated-conformance              Conformance,
        server-max-receive-pdu-size         Unsigned16,
        vaa-name                            ObjectName
  }
```

The value of the vaa-name in case of LN referencing is a dummy value 0x0007. In case of SN referencing, its value is the base_name of the Association SN object, 0xFA00.

In COSEM, the quality-of-service parameter is not used. The server shall accept any value and process the xDLMS-Initiate.request without considering the value of this parameter.

```
ConfirmedServiceError::=        CHOICE
{
        -- tag 0 is reserved
        initiateError           [1]     ServiceError,
        getStatus               [2]     ServiceError,
        getNameList             [3]     ServiceError,
        .                       .       .
        terminateUpLoad         [19]    ServiceError
}
```

where ServiceError is as follows:

```
ServiceError    ::=     CHOICE
{
        .                       .       .
        initiate                [6]     IMPLICIT ENUMERATED
        -- initiate service error
        {
                other                           (0),
                DLMS-version-too-low            (1), -- proposed DLMS version too low
                incompatible-conformance        (2), -- proposed services not sufficient
                PDU-size-too-short              (3), -- proposed PDU size too short
                refused-by-the-VDE-Handler      (4)  -- vaa creation impossible or not allowed
        }
        .       .       .
}
```

NOTE See also Annex A.

8.5 The conformance block

In order to enable optimized COSEM server implementations, a conformance block with extended functionality is added. The COSEM conformance block can be distinguished from the standard conformance block by its tag “APPLICATION 31”.

```
Conformance  ::= [APPLICATION 31] IMPLICIT BIT STRING (SIZE(24))
 {
-- the bit is set when the corresponding service or functionality is available
                  reserved (0)                        (0),
                  reserved (0)                        (1),
                  reserved (0)                        (2),
                  read                                (3),
                  write                               (4),
                  unconfirmed-write                   (5),
                  reserved (0)                        (6),
                  reserved (0)                        (7),
                  attribute0-supported-with-SET       (8),
                  priority-mgmt-supported             (9),
                  attribute0-supported-with-GET       (10),
                  block-transfer-with-get             (11),
                  block-transfer-with-set             (12),
                  block-transfer-with-action          (13),
                  multiple-references                 (14),
                  information-report                  (15),
                  reserved (0)                        (16),
                  reserved (0)                        (17),
                  parameterized-access                (18),
                  get                                 (19),
                  set                                 (20),
                  selective-access                    (21),
                  event-notification                  (22),
                  action                              (23)
  }
```

The parameterized access (as additional variant of the VariableAccessSpecification) provides the ReadRequest or the WriteRequest service with the capability to transport additional parameters.

Parameterized access is introduced by adding the following access method (compare Annex A of IEC 61334-4-41, p. 221):

```
Variable-Access-Specification:= CHOICE {
                                        ...   [2]...
                                        ...   [3]...
         parameterized-access                 [4] IMPLICIT SEQUENCE{
                  variable-name                       ObjectName,
                  selector                            Integer,
                  parameter                           Data
         }
  }
```

The meaning of the selector and of the access parameter depends on the referenced variable. It is defined in the corresponding COSEM IC specification.

8.6 Definition of APDUs for data communication

8.6.1 COSEM APDUs using logical name referencing

```
-- 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-With-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[28]
  }

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-With-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
```

[28] There should be one method-invocation-parameters parameter corresponding to each method-Identifier. When the invoked method – identified by the method-identifier – does not require additional parameters, the corresponding data in the method-invocation-parameters **SEQUENCE OF** should be present as null_data.

```
  {
        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
  }

EVENT-NOTIFICATION-Request         :: = SEQUENCE
  {
        time                                   Cosem-Date-Time OPTIONAL,
        cosem-attribute-descriptor             Cosem-Attribute-Descriptor,
        attribute-value                        Data
  }


EXCEPTION-Response                 ::=SEQUENCE
        {
              state_error     [0],
              service_error   [1]
        }
        state_error           IMPLICIT ENUMERATED
              {
                    service_not_allowed           [1],
                    service_unknown               [2]
              }
        service_error           IMPLICIT ENUMERATED
                {
                        operation_not_possible          [1]
                        service_not_supported           [2]
                        other_reason                    [3]
                }
```

8.6.2 DLMS APDUs using short name referencing

-- APDUs using short name refencing

```
ReadRequest  ::= SEQUENCE OF Variable-Access-Specification

ReadResponse            ::= SEQUENCE OF CHOICE
  {
        data                    [0]     Data,
        data-access-error       [1]     IMPLICIT Data-Access-Result
  }

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
  }

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
  }
```

Annex A
(normative)

The xDLMS application service element

A.1 General

The main objective of the COSEM approach is to provide a business domain oriented interface object model for metering devices and systems while keeping backward compatibility to the existing DLMS standard. To meet these objectives, COSEM includes an evolution of DLMS. Remaining fully compliant to the DLMS standard, COSEM provides a more metering specific view of the meter through the COSEM interface objects.

The xDLMS service element of the COSEM application layer is based on DLMS as specified in IEC 61334-4-41.

A.2 DLMS compliance

A COSEM data exchange session always starts with the AA establishment. This is always initiated by the client. The DLMS services used for accessing attributes and methods of COSEM interface objects are negotiated between the client and the server with the help of the xDLMS-Initiate service during the association establishment. If the response is positive, the AA is established within the given COSEM application context and xDLMS context.

In addition, COSEM specifies a new conformance block extending the number of available DLMS services, see 8.5.

A.3 Extensions to DLMS for COSEM

For the purposes of COSEM, some extensions of the DLMS standard are necessary. They define added functionality i.e. existing functionality is not modified. The extensions are made in such a way, that there is no conflict with the existing DLMS standard.

A.3.1 Additional services

In order to be able to reference attributes and methods of COSEM interface objects using Logical names, the following new services are defined:

- GET – used to retrieve the value of attributes of COSEM interface objects;
- SET – used to set the value of attributes of COSEM interface objects;
- ACTION – used to invoke methods of COSEM interface objects;
- EventNotification – used to send an unsolicited message from the server to the client.

These services are defined in 6.5.3 and in 6.6.3.2 respectively. The corresponding APDUs are defined in 8.6.1.

A.3.2 Additional data types

These are defined in 8.3.

A.3.3 The conformance block

In order to enable optimised COSEM server implementations a conformance block with extended functionality has been added. See 8.5. The COSEM conformance block can be distinguished from the DLMS standard conformance block by its tag "Application 31".

The DLMS application context, using this conformance block is negotiated using the modified xDLMS-Initiate/DLMS-Response services.

A.3.4 DLMS version number

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

A.3.5 Other necessary modifications

The following modifications are necessary to clarify the meaning of the maximum PDU size usable by the client and the server.

IEC 61334-4-41, page 61, Table 3:

was:	new:
Proposed Max PDU Size	Client Max Receive PDU Size

was:	new:
Negotiated Max PDU Size	Server Max Receive PDU Size

IEC 61334-4-41, page 63, 5th paragraph:

was:	new:
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 must be large enough to always permit the Initiate Error PDU transmission.	The Client Max Receive PDU Size parameter, of type Unsigned16, contains the maximum length expressed in bytes for a DLMS PDU that the server may send. The client will discard any received PDUs that are longer than this maximum length. The value must be large enough to always permit the Application Association Response APDU transmission. Values below 12 are reserved. The value 0 indicates that the there is no limit on the PDU size.

IEC 61334-4-41, page 63, last paragraph:

was:	new:
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 the there is no limit on the PDU size.

Annex B
(normative)

Using the COSEM Application Layer in various communication profiles

B.1 Communication profile specific elements

B.1.1 General

As explained in Clause 4, the COSEM interface model for energy metering equipment, specified in IEC 62056-62 has been designed for use with a variety of communication profiles for exchanging data via various communication media.

As shown on Figure 3, in each such profile, the application layer is the COSEM Application Layer, providing the client-server type xDLMS services to access the 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 application layer services to the service set provided and used by the supporting layer;
- communication profile specific parameters of the COSEM application layer services;
- specific considerations/constraints for using certain services within a given profile.

B.1.2 Targeted communication environments

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

B.1.3 The structure of the profile

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

B.1.4 Identification and addressing scheme

As described in 4.5 of IEC 62056-62, metering equipment are modeled in COSEM as physical devices, containing one or more logical devices. In the COSEM Client/Server type communications model data exchange takes place between a COSEM client AP and a COSEM Logical Device, playing the role of the server AP.

In order to be able to establish the required AA and then exchanging data with the help of the supporting lower layer protocols, the Client- and Server APs must be able to be identified and addressed, according to the addressing rules of a communication profile. At least the following elements need to be identified/addressed:

- Physical devices hosting clients and servers;
- Client and Server APs;

The Client and Server applications need also to identify the AAs.

Physical and logical addressing schemes, and AA identification is specific to the profile used. This part specifies these schemes for the given profile.

B.1.5 Supporting layer services and service mapping

In each communication profile, the COSEM application layer 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 application layer, which is using these services:

- to establish, release and abort AAs; and
- to exchange data: APDUs, representing xDLMS Services.

This part describes the "service mapping": it specifies how the application layer is using the specific set of a lower layer profile to provide the COSEM services within the given profile. This part is generally defined using Message Sequence Charts, showing the sequence of the events following the invocation of services by the AP.

B.1.6 Communication profile specific parameters of the COSEM application layer services

In COSEM, only the COSEM-OPEN services have communication profile specific parameters. Their values and use are defined as part of the communication profile specification.

B.1.7 Specific considerations/constraints using certain services within a given profile

The availability and the protocol of some of the services may depend on the communication profile. These elements are specified as part of the communication profile specification.

B.2 The 3-layer, connection-oriented, HDLC based communication profile

B.2.1 Targeted communication environments

The 3-layer, connection-oriented, HDLC based profile is suitable for local data exchange with metering equipment via direct connection, or remote data exchange via the PSTN or GSM networks.

B.2.2 The structure of the profile

B.2.2.1 The protocol layers

This profile is based on the simplest, three-layer (collapsed) OSI protocol architecture. It contains three protocol layers, as follows:

- the COSEM application layer;
- the data link layer based on the HDLC standard;
- the physical layer.

B.2.2.2 The data link layer

The data link layer is based on the ISO/IEC 13239 standard. The second edition of the standard includes a number of enhancements compared to the original HDLC standard, for example in the areas of addressing, error protection, and segmentation. The third edition incorporates a new frame format, which meets the requirements of the environment found in telemetry applications for electricity metering and similar industries.

For the purpose of this profile, the following selections have been made:

- unbalanced connection-mode data link operation[29];
- two-way alternate data transfer;
- the selected HDLC class of procedure is UNC, extended with UI frames;
- frame format type 3;
- non-basic frame format transparency.

The HDLC based data link layer is defined in IEC 62056-46.

B.2.2.3 The physical layer

The physical layer for this profile is defined in IEC 62056-42. It provides the specification of services and procedures for connection-oriented asynchronous data exchange.

To allow using a wide variety of media, this standard does not specify the physical layer signals and their characteristics. However, the following assumptions are made:

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

IEC 62056-42 provides a protocol identification service and gives an example for using the physical layer for data exchange through the Public Switched Telephone Network (PSTN) with intelligent Hayes modems. The use of the physical layer for the purposes of direct local data exchange using an optical port or a current loop physical interface is specified in IEC 62056-21.

B.2.3 Identification and addressing scheme

The HDLC based data link layer provides services to the COSEM Application Layer at Data Link SAPs – also called the Data Link or HDLC Address.

On the client side, only the client AP needs to be identified. The addressing of the physical device hosting the client APs is done by the physical layer (e.g. by using phone numbers).

On the server side, several physical devices may share a common physical line (multi drop configuration). In the case of direct connection this may be a current loop as specified in IEC 62056-21, or in the case of remote connection, several physical devices may share a single telephone line. Therefore, both the physical devices and the logical devices hosted by the physical devices need to be identified. This is done using the HDLC addressing mechanism as described in 6.4.2 of IEC 62056-46:

- the physical devices are identified by the lower HDLC address;
- the logical devices within a physical device are identified by the upper HDLC address;
- a COSEM AA is identified by a doublet, containing the identifiers of the two APs participating in the AA.

[29] In DLMS/COSEM, the primary station corresponds to the client application, and the secondary station corresponds to the server application.

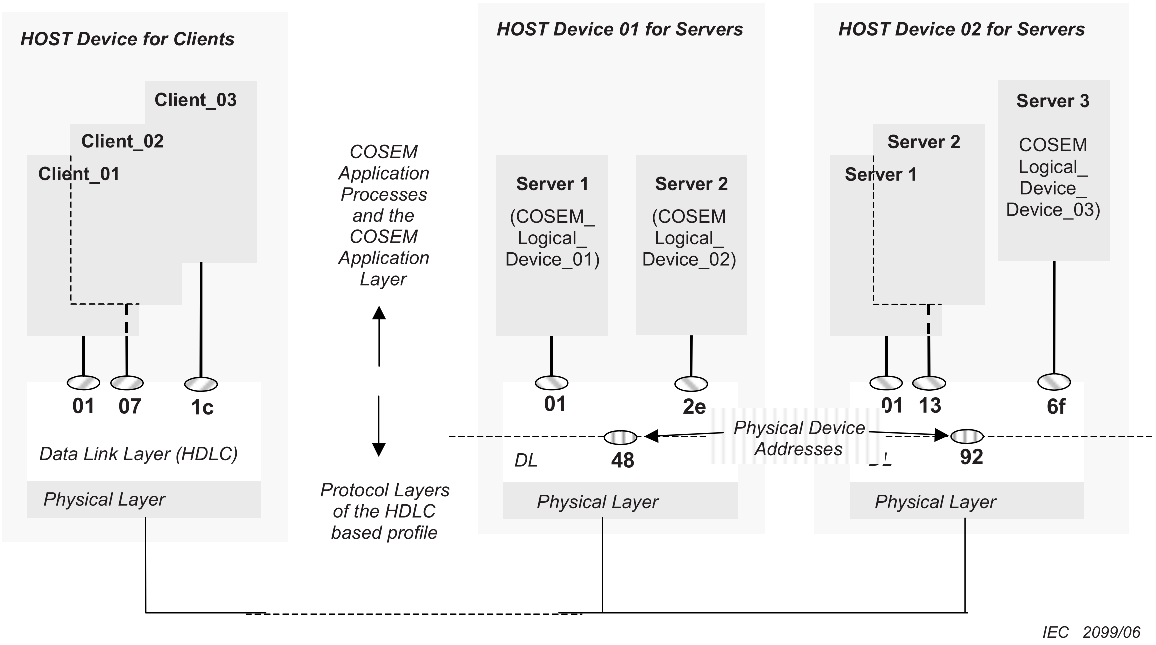

HOST Device for Clients
Client_03
Client_02
Client_01
01
07
1c
Data Link Layer (HDLC)
Physical Layer
COSEM Application Processes and the COSEM Application Layer
Protocol Layers of the HDLC based profile
HOST Device 01 for Servers
Server 1
(COSEM_ Logical_ Device_01)
Server 2
(COSEM Logical_ Device_02)
01
2e
DL
48
Physical Layer
Physical Device Addresses
HOST Device 02 for Servers
Server 3
COSEM Logical_ Device_ Device_03)
Server 2
Server 1
01
13
6f
DL
92
Physical Layer
IEC 2099/06

Figure B.1 – Identification/addressing scheme in the 3-layer, connection-oriented, HDLC based communication profile

For example, an AA between Client_01 (HDLC address = 01) and Server 2 in Host Device 02 (HDLC address = 1392) is identified by the doublet {01, 1392}. Here, “13” is the upper HDLC address and “92” is the lower HDLC address. All values are hexadecimal.

This scheme ensures that a particular COSEM AP (client or server) may have more than one simultaneous AA without ambiguity.

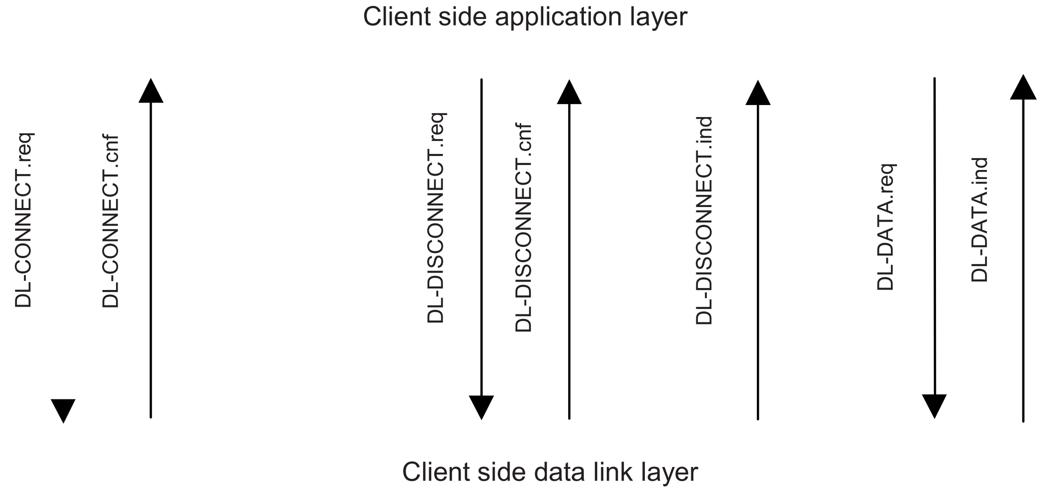
B.2.4 Supporting layer services and service mapping

The HDLC based data link layer provides services for

- data link layer connection management;
- connection-oriented data communication;
- connectionless data communication.

B.2.4.1 Data link layer services provided at the client side

Figure B.2 summarizes the data link layer services provided to and used by the COSEM client application layer.



IEC 2100/06

Figure B.2 – Data link layer services provided to and used by the client COSEM application layer

For some services, the correspondence between an application layer (ASO) service invocation and the supporting data link layer service invocation is straightforward. For example, invoking a COSEM GET.request service directly implies the invocation of a DL-DATA.request service.

On the other hand, for some other services direct service mapping cannot be established. For example, the invocation of a COSEM-OPEN.request service with Service_class == Confirmed launches a series of actions, starting with the establishment of the lower layer connection with the help of the DL-CONNECT service, and then sending out the AARQ APDU via this newly established connection using a DL-DATA.request service. Examples for service mapping are given in this standard.

B.2.4.2 Data link layer services provided at the server side

Figure B.3 summarizes the data link layer services provided to and used by the COSEM server application layer.

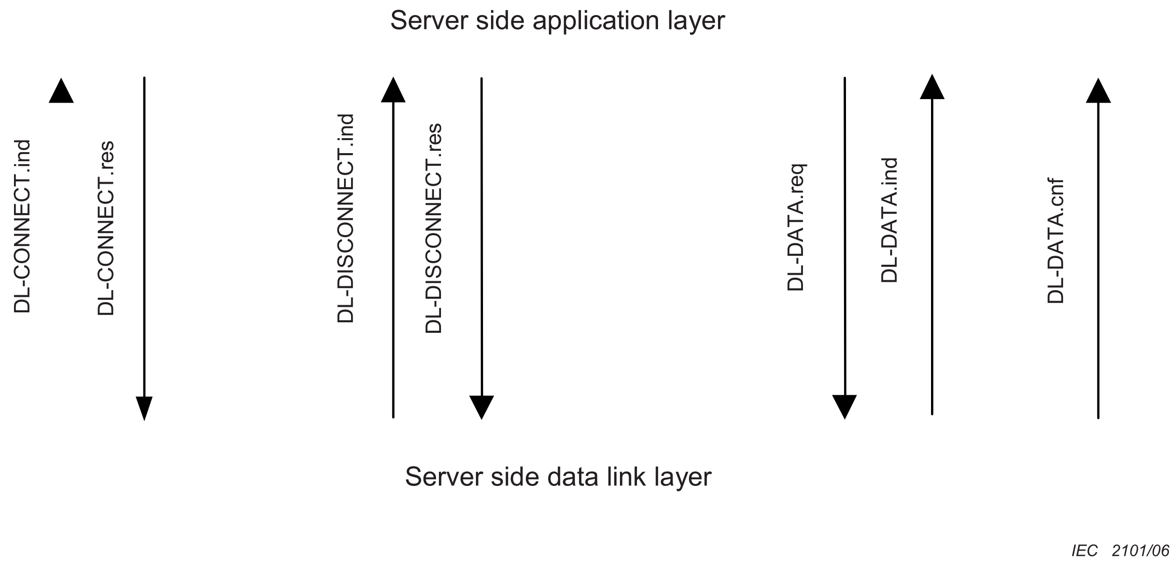


Figure B.3 – Data link layer services provided to and used by the server COSEM application layer

The situation is similar to the one at the client side. For some services, the correspondence between an application layer (ASO) service invocation and the supporting data link layer service invocation is straightforward. For other services, this direct service mapping cannot be established.

In this profile, the data link layer supports transfer of long data from the server to the client, in a transparent manner to the application layer, using the segmentation feature of the HDLC based data link layer. To support this feature, a local DL-DATA.confirm service is available on the server side.

B.2.5 Communications profile specific service parameters of the COSEM application layer services

The only application layer service, which has communication specific parameters, is the COSEM-OPEN service.

The address information, required to establish a data link layer connection and then an AA, is carried by the Protocol_Connection_Parameters parameter of the COSEM-OPEN service. This information includes the following data:

• Protocol (Profile) Identifier	3-Layer, connection-oriented, HDLC based;
• Server_Lower_MAC_Address	(COSEM Physical Device Address);
• Server_Upper_MAC_Address	(COSEM Logical Device Address);
• Client_MAC_Address;	
• Server_LLC_Address;	
• Client_LLC_Address	

Any server (destination) address parameter may contain special addresses (All-station, No-station, etc.).

For more information, see IEC 62056-46.

B.2.6 Specific considerations/constraints

B.2.6.1 Application association types, confirmed and unconfirmed service invocations and frame types used

The following table summarizes the rules for establishing confirmed and unconfirmed AAs, the type of data communication services, which can be used in these associations and the HDLC frame types used for carrying APDUs.

Table B.1 – Application associations and data exchange in the 3-layer, connection-oriented, HDLC based profile

Application association establishment				Data exchange	
Protocol connection parameters	COSEM-OPEN service class	Use	Type of established application association	Service class	Use
Id: HDLC LLC and MAC addresses	Confirmed	1/ Connect data link layer 2/ Exchange AARQ/AARE APDUs transported in “I” frames	Confirmed	Confirmed	“I” frame
				Unconfirmed	“UI” frame
	Unconfirmed	Send AARQ in a “UI” frame	Unconfirmed	Confirmed (not allowed)	-
				Unconfirmed	“UI” frame

This table clearly shows one of the specific features of this profile:

- in confirmed AAs, APDUs corresponding to confirmed service invocations (e.g. a SET.request service invocation with Service_class == confirmed) are sent using connection-oriented data services of the supporting data link layer: the APDU is carried in “I” frames;
- if, within the same AA a data communication service is invoked in a non-confirmed manner (Service_Class == Unconfirmed) then the corresponding APDU is sent using connectionless data services of the supporting data link layer: the APDU is sent in a “UI” type frame;
- consequently, bit 6 of the Invoke_id_And_Priority field is not relevant in this profile. See also 8.3.

B.2.6.2 Correspondence between application associations and data link layer connections, association release

In this profile, a confirmed AA is bound to a supporting data link layer connection, in a one-to-one basis. Consequently:

- establishing a confirmed AA shall imply the establishment of a connection between the client and server data link layers;
- a confirmed AA in this profile can be non-ambiguously released by disconnecting the corresponding data link layer connection.

On the other hand, in this profile establishing a non-confirmed AA does not need any lower layer connection: consequently, once established, non-confirmed AAs with servers not supporting releasing AAs using the RLRQ APDU (see 6.5) cannot be released.

B.2.6.3 Service parameters of the COSEM-OPEN/RELEASE/ABORT services

Thanks to the possibility to transparently transport higher layer related information within the SNRM and DISC HDLC frames, this profile allows the use of the optional “User_Information” parameter of the COSEM-OPEN.request/.indication and COSEM-RELEASE.request /.indication services.

- The User_Information parameter of the COSEM-OPEN.request service (see 6.5.1.2) shall be inserted into the "User data subfield" of the SNRM HDLC frame, sent during the data link connection establishment;
- On the server side, if the SNRM frame received contains a "User data subfield", the contents of this field shall be transmitted to the server AP via the User_Information parameter of the COSEM-OPEN.indication service (see 6.6.1.2);
- The User_Information parameter of the COSEM-RELEASE.request service, when it is present, (see 6.5.2.2) shall be inserted into the "User data subfield" of the DISC HDLC frame, sent during disconnecting the data link connection;
- If the DISC frame received by the server contains "User data subfield", the contents of this field shall be transmitted to the server AP via the User_Information parameter of the COSEM-RELEASE.indication service, see 6.6.2.1;
- The User_Information parameter of the COSEM-RELEASE.response service, when it is present, (see 6.6.2.2) shall be inserted into the "User data subfield" of the UA or DM HDLC frame, sent in response to the DISC frame;
- If the UA or DM frame received by the client contains "User data subfield", the contents of this field shall be transmitted to the client AP via the User_Information parameter of the COSEM-RELEASE.confirm service, see 6.6.2.2;

In addition, for the COSEM-ABORT.indication service, the following rule applies:

- the Diagnostics parameter of the COSEM-ABORT.indication service (see 6.5.2.4 and 6.6.2.3) may contain an unnumbered send status parameter. This parameter indicates whether, at the moment of the physical abort indication, the data link layer has or does not have a pending Unnumbered Information message (UI). The type and the value of this parameter is a local issue, thus it is not within the scope of this standard;

B.2.6.4 EventNotification service and protocol

In this profile, an event is always reported by the server management logical device (mandatory, reserved upper HDLC address 0x01) to the client management AP (mandatory, reserved HDLC address 0x01).

The EVENT-NOTIFICATION-Request APDU shall be sent out using the connectionless data services, i.e. in an UI frame, at the first opportunity when the server side data link layer receives the right to talk. The APDU shall fit into one HDLC frame.

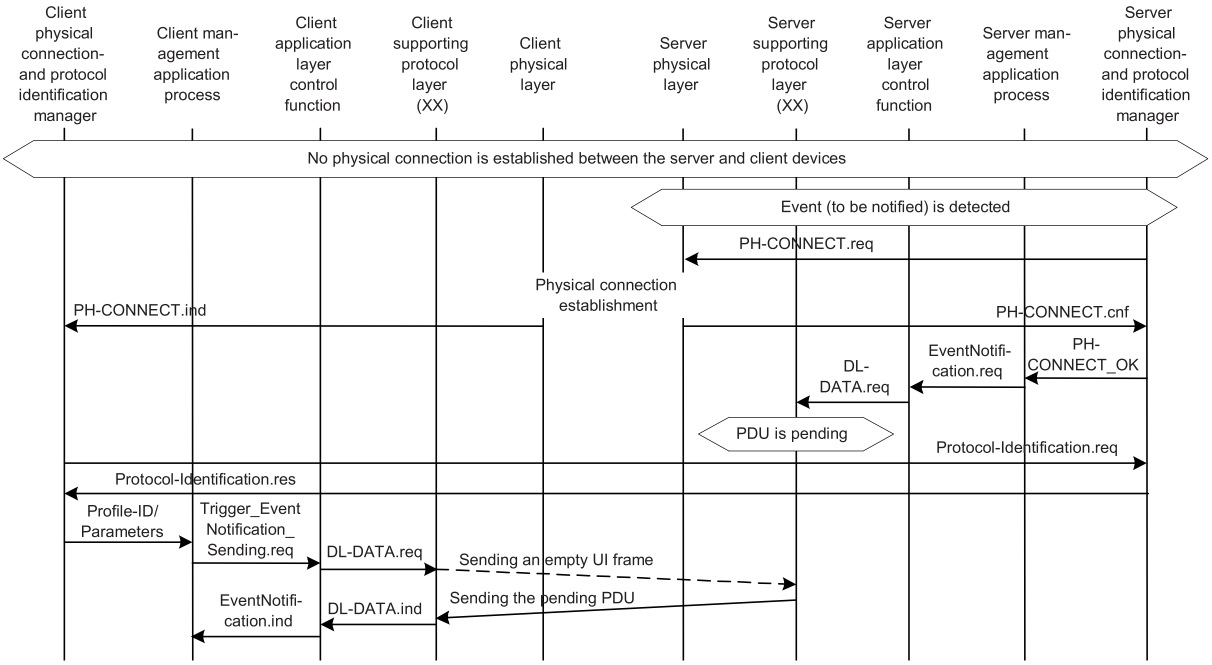
In order to be able to send out the APDU, a physical connection between the physical device hosting the server and a client device must exist, and the server side data link layer needs to receive the token from the client side data link layer.

If there is a data link connection between the client and the server when the event occurs, the server side data link layer may send out the PDU carrying the EVENT-NOTIFICATION-Request APDU, following the reception of an I frame, a UI frame or an RR frame from the client. These possibilities are defined in IEC 62056-46.

If there is no physical connection when the event occurs (but this connection to a client device can be established) then the first step is to establish this physical connection.

NOTE Physical connection cannot be established when the server has only a local interface (e.g. an optical port as defined in IEC 62056-21) and the hand-held terminal, running the client application is not connected, or the server has a PSTN interface, but the telephone line is not available. Handling such cases is implementation specific.

Following the establishment of the physical connection, the client has to trigger the sending of the EVENT-NOTIFICATION-Request APDU using the Trigger_EventNotification_Sending.request service. The MSC shown on Figure B.4 represents this case.


Client physical connection- and protocol identification manager
Client man-agement application process
Client application layer control function
Client supporting protocol layer (XX)
Client physical layer
Server physical layer
Server supporting protocol layer (XX)
Server application layer control function
Server man-agement application process
Server physical connection- and protocol identification manager
No physical connection is established between the server and client devices
Event (to be notified) is detected
PH-CONNECT.req
Physical connection establishment
PH-CONNECT.ind
PH-CONNECT.cnf
PH-CONNECT_OK
EventNotifi-cation.req
DL-DATA.req
PDU is pending
Protocol-Identification.req
Protocol-Identification.res
Profile-ID/ Parameters
Trigger_Event Notification_ Sending.req
DL-DATA.req
Sending an empty UI frame
Sending the pending PDU
EventNotifi-cation.ind
DL-DATA.ind

IEC 2102/06

Figure B.4 – Example: EventNotificaton triggered by the client

The first action of the server is to establish a physical connection to the client.

NOTE This physical connection establishment is done outside the protocol stack.

Successful physical connection establishment is reported at both sides to the physical connection and protocol identifier manager process. At the server side, this shall indicate to the COSEM AP, that the EventNotification.request service can be invoked now. When it is done, the server application layer shall build an EVENT-NOTIFICATION-Request APDU and shall invoke the connectionless DL-DATA.request service of the data link layer with the data parameter carrying the APDU.

At this moment, the data link layer may not be able to send this PDU immediately, thus it will be stored in the data link layer, waiting to be sent (pending).

When the client detects a successful physical connection establishment – and as there is no other reason to receive an incoming call – it shall suppose that this call is originated by a remote server intending to send an EventNotification message.

The client, at this moment may not know the protocol stack used by the calling server. Therefore, it has to identify it first using the optional protocol identification service described in IEC 62056-42. This is shown as a “Protocol-Identification.request” and a “Protocol-Identification.response” messages in Figure B.4. After the identification of the protocol stack used by the server, the client is able to instantiate the right protocol stack.

The client AP shall then invoke the Trigger_EventNotification_Sending.request service of the client application layer (see 6.5.4.2). Upon invocation of this service, the application layer shall invoke the connectionless DL-DATA.request service of the data link layer with empty data, and the data link layer shall send out an empty UI frame with the P/F bit set to TRUE, giving the permission to the server side data link layer to send the pending PDU.

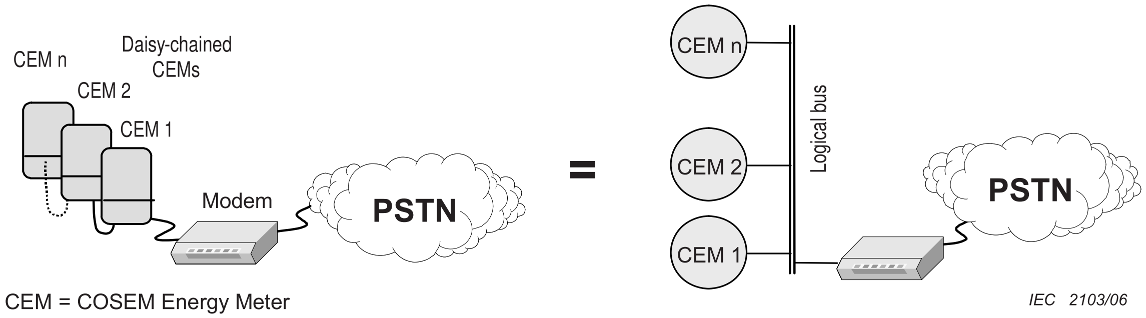
The received Event-Notification-Request APDU shall be indicated to the client AP as an EventNotification.indication. At this moment, the client is notified about the event, then the sequence is completed.

B.2.6.5 Transporting long messages

In this profile, the data link layer provides a method for transporting long messages in a transparent manner for the application layer. This is described in 6.4.4.5 of IEC 62056-46.

B.2.6.6 Supporting multi-drop configurations

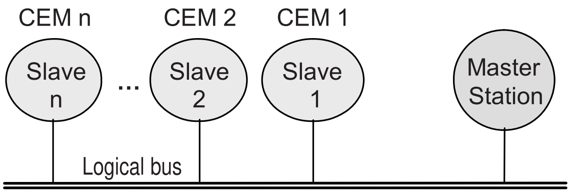
For data exchange with metering equipment, a multi-drop arrangement is often used allowing a data collection system to exchange data with multiple metering equipment, using a shared communication resource like a telephone modem. Various physical arrangements are available, like a star, daisy chain or a bus topology. These arrangements can be modeled with a logical bus, to which the metering equipment and the shared resource are connected, see Figure B.5.



IEC 2103/06

Figure B.5 – Multi-drop configuration and its model

As collision on the bus must be avoided but a protocol, controlling access to the shared resource is not available, access to the bus must be controlled by external rules. In most cases, a Master-Slave arrangement is used, where the metering equipment is the Slaves. Slave devices have no right to send messages without first receiving an explicit permission from the Master.



IEC 2104/06

Figure B.6 – Master/ Slave operation on the multi-drop bus

In COSEM, data exchange takes place based on the Client-Server model. Physical devices are modeled as a set of logical devices, and these logical devices are acting as servers, providing responses to requests.

Obviously, the Master Station of a multi-drop configuration is located at the other side of the communication channel and it is acting as the client, sending requests and expecting responses.

The client may send requests at the same time to multiple servers, if no response is expected (multicast or broadcast).

If the client expects a response, it must send the request to a single server, giving also the right to talk. It has to wait then for the response before it may send a request to another server and with this, giving the right to talk. Arbitration of access to the common bus is thus controlled in a time-multiplexing fashion.

Messages from the client to the servers must contain addressing information. In the 3-layer, connection-oriented, HDLC based profile, this is ensured by using HDLC addresses. If a multi-drop arrangement is used, the HDLC address has to be split into two parts: the lower HDLC address is used to address physical devices, and the upper HDLC address is used to address logical devices within the physical device. Both the lower and the upper address may contain a broadcast address. For more detail, see IEC 62056-46.

In order to be able to report events, COSEM meters may initiate a connection to the client, using the non-client-server type EventNotification/InformationReport services. As events in several or all meters connected to a multidrop may occur simultaneously – for example in the case of a power failure, they may initiate a call to the client simultaneously.

For such cases, two problems have to be handled:

- collision on the logical bus: For the reasons explained above, several physical devices may try to access the shared resource (e.g. sending AT commands to the modem) simultaneously. This would result in a collision on the bus. Such situations must be handled by the manufacturers;
- identification of the originator of the event report: This is possible by using the CALLING Physical Device Address, as described in 6.4.4.8 of IEC 62056-46.

B.3 The TCP-UDP/IP based communication profiles (COSEM_on_IP)

B.3.1 Targeted communication environments

The TCP-UDP/IP based profiles are suitable for remote data exchange with metering equipment via IP enabled Networks (e.g. Internet), using various communication networks, such as Local Area Networks (Ethernet, Bluetooth, WiFi, etc.) or public or private Wide Area Networks, such as PSTN, GPRS, etc.

B.3.2 The structure of the profile(s)

B.3.2.1 The protocol layers

In the TCP-UDP/IP based profiles the COSEM Application layer uses the services of one of the Internet transport layers (TCP or UDP) via a wrapper, which, in their turn, uses the services of the Internet Protocol (IP) network layer to communicate with other nodes connected to this abstract network.

The TCP-UDP/IP layers are implemented on a wide variety of real networks, which, just with the help of this IP Network abstraction, can be seamlessly interconnected to form Intra- and Internets using any set of lower layers supporting the Internet Protocol.

The COSEM Application layer in this environment can be considered as another Internet standard application protocol, which may co-exist with other Internet application protocols, like the well-known FTP, HTTP, etc. services, as shown in Figure B.7.

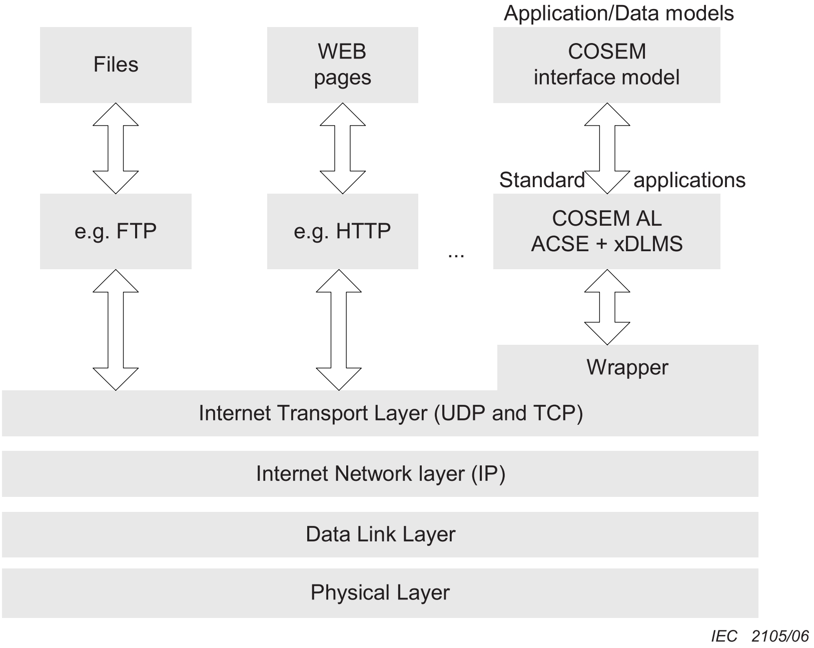


Figure B.7 – COSEM as a standard Internet application protocol

The TCP-UDP/IP based communication profiles for COSEM consist of five protocol layers:

- the COSEM Application Layer;
- the COSEM Transport Layer, as defined in: It is based on the connection-oriented Transmission Control Protocol (TCP) or the connectionless User Datagram Protocol (UDP) Internet transport protocols and includes a wrapper;
- network Layer: The Internet Protocol (IP);
- data Link Layer: Any data link protocols supporting IP (PPP, Ethernet, etc.);
- physical Layer: any physical layer supported by the data link layers chosen.

B.3.2.2 The COSEM transport layer

The COSEM transport layer of this profile is based on the connection-oriented (TCP, STD0007) and connectionless (UDP, STD0006) Internet transport protocols. It is specified in IEC 62056-47.

B.3.2.3 The IPv4 network layer

In the TCP-UDP/IP based communication profile the network layer is the Internet protocol, as specified in the Internet standard STD0005. The COSEM communication profile makes use of this protocol according to this specification: no special rules or restrictions apply.

B.3.2.4 Lower protocol layers

One of the reasons of the success of the Internet protocols is just its federating force. Practically any data communication networks, Wide Area Networks (such as ISDN, GPRS, ATM and Frame Relay networks), data communications on the circuit switched PSTN networks (dial-up IP), as well as Local Area Networks (such as Ethernet, Wireless LAN, Token Ring, FDDI, Bluetooth, etc.) support TCP-UDP/IP networking.

Figure B.8 shows a set of examples – far-from-being-complete – for such communication networks and for the lower layer protocols used in these networks.

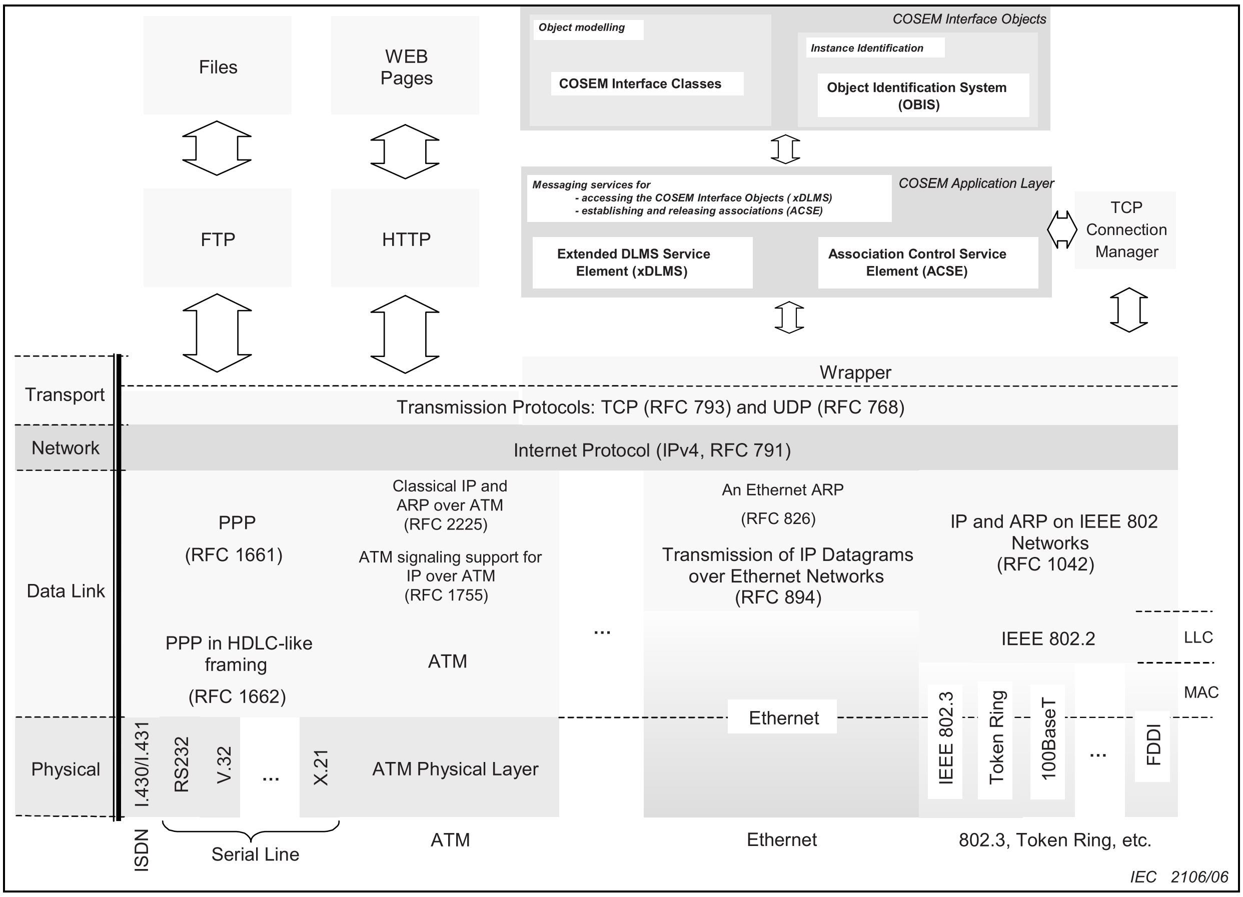

Files
WEB Pages
Object modelling
COSEM Interface Objects
COSEM Interface Classes
Instance Identification
Object Identification System (OBIS)
FTP
HTTP
Messaging services for
- accessing the COSEM Interface Objects (xDLMS)
- establishing and releasing associations (ACSE)
COSEM Application Layer
Extended DLMS Service Element (xDLMS)
Association Control Service Element (ACSE)
TCP Connection Manager
Wrapper
Transport
Transmission Protocols: TCP (RFC 793) and UDP (RFC 768)
Network
Internet Protocol (IPv4, RFC 791)
Data Link
PPP
(RFC 1661)
Classical IP and ARP over ATM (RFC 2225)
ATM signaling support for IP over ATM (RFC 1755)
An Ethernet ARP
(RFC 826)
Transmission of IP Datagrams over Ethernet Networks (RFC 894)
IP and ARP on IEEE 802 Networks (RFC 1042)
PPP in HDLC-like framing
(RFC 1662)
ATM
...
IEEE 802.2
LLC
MAC
Ethernet
Physical
I.430/I.431
RS232
V.32
...
X.21
ATM Physical Layer
IEEE 802.3
Token Ring
100BaseT
...
FDDI
ISDN
Serial Line
ATM
Ethernet
802.3, Token Ring, etc.
IEC 2106/06

Figure B.8 – Examples for lower-layer protocols in the TCP-UDP/IP based profiles

Using the TCP-UDP/IP profile, COSEM can be used practically on any existing communication network.

B.3.3 Identification and addressing scheme

Although real-world devices even in the Internet environment are connected to real-world physical networks, at a higher abstraction (and protocol) level it can be considered as if these devices would be connected to a virtual – IP – network. On this virtual network, each device has a unique address, called IP address, which non-ambiguously identifies the device on this network.

Any device connected to this virtual IP network can send (a) message(s) to any other connected device(s) using only the IP Address to designate the destination device, without being concerned about the complexity of the whole physical network. Specific characteristics – the data transmission medium, the media access strategy, and the specific data-link addressing/identification scheme – of the particular physical network(s) participating in the route between the source and the destination device are hidden for the sender device. These elements are handled by intermediate network devices, called routers.

Therefore, in the TCP-UDP/IP based profiles COSEM physical devices are non-ambiguously identified by their network – IP – address.

The identification of an AP – a COSEM client AP or a COSEM Server Logical Device – within a physical device requires an additional address.

Both TCP and UDP provide additional addressing capability at the transport level, called *port*, to distinguish applications running on the same physical device.

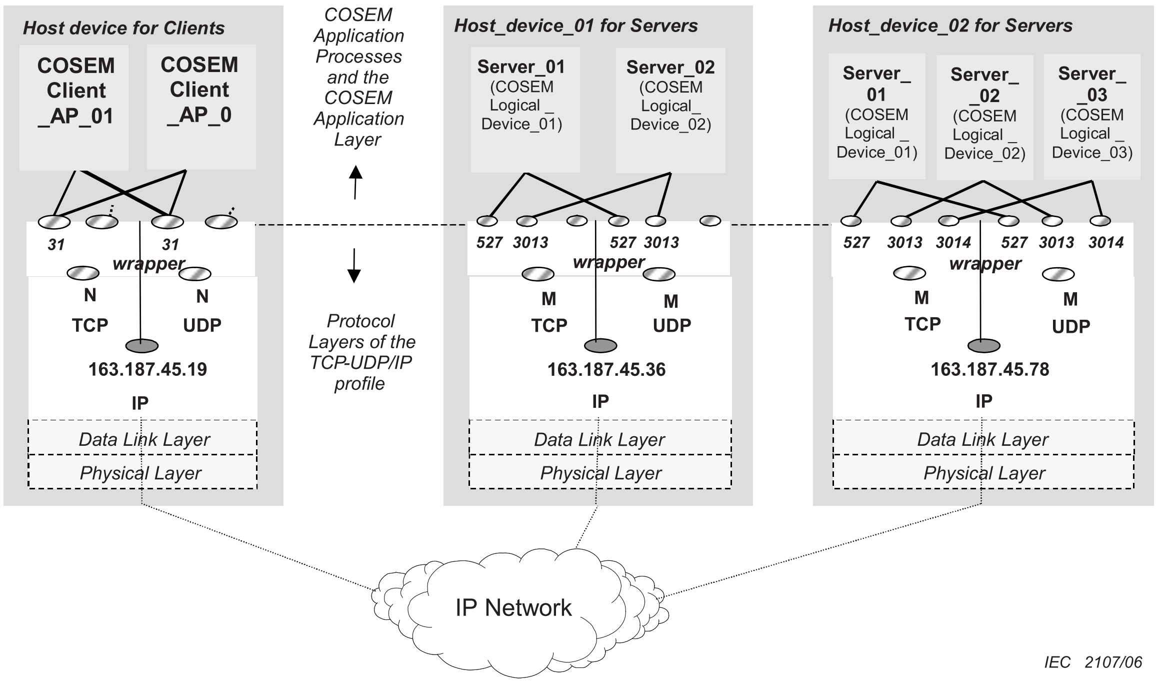
The Application Layer is listening only on one TCP or UDP port for exchanging messages between client and server applications. As in a single physical device, several APs – client AP or server logical devices – may be present, an additional addressing capability is needed.

This is provided by the wrapper sub-layer (see IEC 62056-47). The wrapper provides an identifier – wPort – similar to the TCP or UDP port numbers, but on the top of these layers. A particular COSEM client AP and/or a particular COSEM logical device in the same physical device (identified by their IP Addresses) can be thus identified by its wPort number.

Therefore, in the TCP-UDP/IP based profile the following identification rules apply:

- COSEM physical devices are identified by their IP address;
- the COSEM Application layer is listening only one one UDP or TCP port;
- COSEM Logical Devices and client APs within their respective host physical devices are identified by their wPort numbers;
- reserved wPort numbers for the Public Client AP and for the Management Logical Device are specified in IEC 62056-47;
- lower layer addresses (SAP-s) are not considered (hidden).

Figure B.9 shows the concept of this identification/addressing scheme.


Host device for Clients
COSEM Client _AP_01
COSEM Client _AP_0
31
31
wrapper
N
N
TCP
UDP
163.187.45.19
IP
Data Link Layer
Physical Layer
COSEM Application Processes and the COSEM Application Layer
Protocol Layers of the TCP-UDP/IP profile
Host_device_01 for Servers
Server_01 (COSEM Logical _ Device_01)
Server_02 (COSEM Logical_ Device_02)
527 3013
527 3013
wrapper
M
M
TCP
UDP
163.187.45.36
IP
Data Link Layer
Physical Layer
Host_device_02 for Servers
Server_ 01 (COSEM Logical _ Device_01)
Server_ _02 (COSEM Logical _ Device_02)
Server_ _03 (COSEM Logical _ Device_03)
527 3013 3014
527 3013 3014
wrapper
M
M
TCP
UDP
163.187.45.78
IP
Data Link Layer
Physical Layer
IP Network

IEC 2107/06

Figure B.9 – Identification/addressing scheme in the TCP-UDP/IP based profile(s)

COSEM AAs are identified by the identifiers of the two end-points. The end-point identifiers themselves are triplets, consisting of the IP Address of the host device, the Transport address (TCP or UDP port number) used for DLMS/COSEM and the wrapper port number (wPort) identifying the COSEM AP.

For example, associations established between Client_AP_01 and Logical_Device_01 in Host_device_01 (AA 1) and Logical_Device_02 in Host_Device_02 (AA2) respectively are identified by

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

NOTE 1 T_N and T_M means the TCP port used for DLMS/COSEM in the client host device and the server host devices respectively.

NOTE 2 In these two AAs, the client side end-point identifer is the same. However, the server side end-point identifiers are different, so the two associations are identified unambiguously and therefore they can be used simultaneously.

B.3.4 Supporting layer services and service mapping

As defined in IEC 62056-47, the COSEM TCP transport layer provides the following services to the service user:

Connection Management (performed by the TCP connection manager AP):

TCP-CONNECT services	-.request,.indication,.response,.confirm
TCP-DISCONNECT services	-.request,.indication,.response,.confirm

Data exchange (performed by the COSEM Application layer; these services can be used only when the TCP connection is established):

TCP-DATA service	-.request,.indication, (. confirm)

The TCP transport layer also provides a TCP-ABORT.indication service to the service user COSEM application layer to indicate the disconnection/disruption of the TCP layer connection.

The UDP transport layer provides only one service to the service user COSEM Application layer: a connectionless, unreliable data delivery service.

UDP-DATA service	-.request,.indication, (.confirm)

NOTE A.confirm service primitive is optional as well for the TCP and for the UDP data services.

Figure B.10 summarizes the COSEM TCP-UDP transport layer services provided to or used by the service user COSEM application layer and TCP connection manager process.

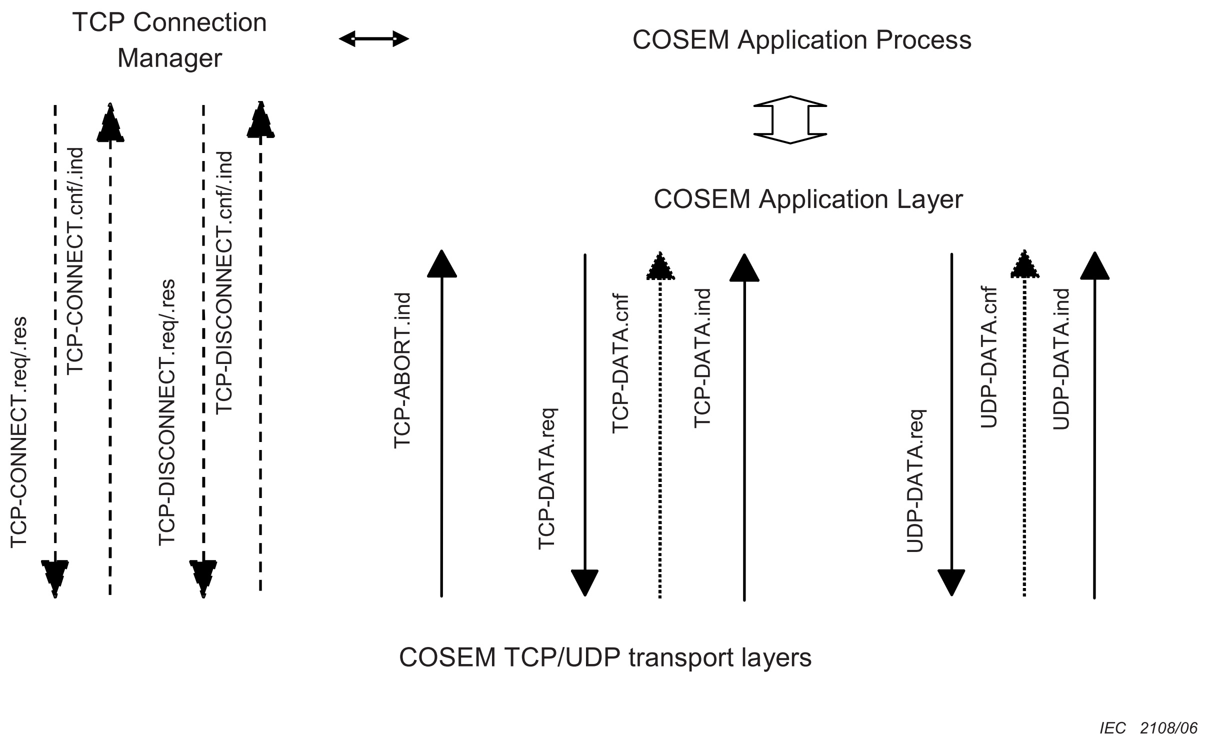


Figure B.10 – Summary of TCP/UDP layer services on the client and server side

For connection management, the COSEM TCP transport layer provides the full set of TCP-CONNECT and TCP-DISCONNECT services at both the Client and the Server sides. The user of these services is not the COSEM Application layer, but the TCP Connection Manager AP. This process is implementation dependent, therefore it is out of the scope of this standard. The only requirements with regard to this process are:

- the TCP connection manager process shall be able to establish the supporting TCP connection without the intervention of the COSEM client or server AP(s);
- the COSEM client and server APs must be able to retrieve the TCP and IP portion of the Protocol_Connection_Parameters parameter from the TCP connection manager before sending/receiving a COSEM-OPEN.request/.indication.

The reason for providing the full TCP connection management service set at both the client and the server sides is, that the establishment and release of the supporting TCP connection may optionally be requested by the Server, too. As in all COSEM profiles, establishment and release of AAs is initiated by the client AP in these profiles as well. This is further discussed in B.3.6.7.

For data exchange, both the Client and the Server application layers use the complete set of service primitives provided by the COSEM TCP-UDP transport layers.

The correspondence between an application layer (ASO) service invocation and the supporting COSEM TCP-UDP layer service invocation is given in IEC 62056-47.

B.3.5 Communication profile specific service parameters of the COSEM application layer services

The only application layer service, which has communication specific parameters, is the COSEM-OPEN service.

The address information, required to identify an AA, is carried by the Protocol_Connection_ Parameters parameter of the COSEM-OPEN service:

Protocol (Profile) Identifier	TCP/IP or UDP/IP;
• Server_IP_Address	COSEM Physical Device Address;
• Server_TCP_or_UDP_Port	The TCP or UDP port used for DLMS/COSEM;
• Server_wrapper_Port	COSEM Logical Device Address;
• Client_IP_Address	COSEM Client’s Physical Device Address;
• Client_TCP_or_UDP_Port	The TCP or UDP port used for DLMS/COSEM;
• Client_wrapper_Port	COSEM Application process (type) identifier.

Any server address parameter may contain special addresses (All-station, No-station, etc.).

For more information, see IEC 62056-47.

B.3.6 Specific considerations/constraints

B.3.6.1 Application association types, confirmed and unconfirmed service invocations and packet types used

The following table shows the rules for establishing confirmed and unconfirmed AAs, the type of data communication services, which can be used in these associations and the transport layer packet types used for carrying APDUs.

Table B.2 –Application associations and data exchange in the TCP-UDP/IP based profile

Application association establishment				Data exchange	
Protocol connection parameters	COSEM-OPEN service class	Use	Type of established application association	Service class	Use
Id: TCP/IP TCP port numbers, IP addresses	Confirmed	1/ Connect TCP layer 2/ Exchange AARQ/AARE APDUs transported in TCP packets	Confirmed	Confirmed	TCP packet
				Unconfirmed	TCP packet
	Unconfirmed	Local negative confirmation	None	-	-
				-	-
Id: UDP/IP UDP port numbers, IP addresses	Confirmed	Exchange AARQ/AARE APDUs transported in UDP datagrams	Confirmed	Confirmed	UDP datagram
				Unconfirmed	UDP datagram
	Unconfirmed	Send AARQ in a UDP datagram	Unconfirmed	Confirmed (not allowed)	-
				Unconfirmed	UDP datagram

In this table, grey areas represent cases which are out of the normal operating conditions: either not allowed or have no useful purpose. According to these:

- it is not allowed to establish a non-confirmed AA using the TCP/IP protocol. It is prevented by the Client application Layer, which shall locally and negatively confirm COSEM-OPEN.requests trying to do that;
- it is not allowed to request an xDLMS service in a confirmed way (Service_class = confirmed) within a non-confirmed AA, established on the top of the UDP layer. This must also be prevented by the Client application layer. Servers, receiving such APDUs shall simply discard them, or, if the feature is implemented, send back the optional EXCEPTION-Response APDU.

B.3.6.2 Releasing application associations: using RLRQ/RLRE is mandatory

Using the A-RELEASE services of the ACSE – by invoking the COSEM-Release.request service with Use_RLRQ_RE == TRUE – in the TCP-UDP/IP based profile is mandatory for the following reasons:

- according to the identification/addressing scheme used in this profile, an AA is identified by two triplets, including the IP Address, the TCP (or UDP) port number and the wPort number. In other words, all AAs within this profile are established using only one TCP (or UDP) port. This means, that disconnecting the TCP connection (this way of releasing AA must also be supported) would release all AAs established. Using the RLRQ/RLRE APDUs allows to release confirmed AAs in a selective way;
- in the TCP-UDP/IP based profile, it is allowed to establish both confirmed and unconfirmed AAs on the connectionless UDP transport layer. The only way to release such associations is the use of the RLRQ/RLRE services.

NOTE In fact, using the RLRQ/RLRE APDUs is specified as optional only to keep backward compatibility with the previous version of the Application Layer standard (IEC 620056-53: 2002), which did not include this possibility.

B.3.6.3 Service parameters of the COSEM-OPEN/RELEASE/ABORT services

The optional User_Information parameters of the COSEM-OPEN/RELEASE services are not supported in this communication profile.

B.3.6.4 xDLMS data communication service related issues/constraints

B.3.6.4.1 Client/Server type services

No specific features/constraints apply related to the use of Client/Server type services.

B.3.6.4.2 The EventNotification Service and the Trigger_EventNotification_Sending service

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

The EVENT-NOTIFICATION-Request APDU may be sent either using the connectionless data services of the COSEM UDP based transport layer or by the connection-oriented data services of the COSEM TCP based transport layer. In this case, a TCP connection has to be built first by the TCP Connection Manager process.

The optional Application_Addresses parameter is present only when the Event Notification.request service is invoked outside an established AA.

B.3.6.5 Transporting long messages

The use of the length bytes in the COSEM TCP-UDP transport layer wrapper element allows sending long messages in a transparent way to the application layer. However, confirmation of reception of the individual fragments is not provided.

B.3.6.6 Multi-drop configuration

Among many other possibilities, the TCP-UDP/IP communication can be used over serial connection links, like the PSTN. In this case, the data link layer protocol to be used is the PPP (Point-to-Point) protocol.

However, multi-drop configurations in this profile are not supported, because with using PPP, addressing is not available. Therefore, a mechanism of avoiding a collision on the logical bus is not available.

B.3.6.7 Allowing the COSEM Server to establish the TCP connection

In COSEM, supporting layer connections are generally established during the application establishment following the invocation of the COSEM-OPEN.request service by the client AP (the physical layer connection must be already established before invoking the COSEM-OPEN.request service). Therefore, linking the process of establishing an AA and connecting the supporting layer is just natural.

However, in some cases it would be useful if the server could also initiate the connection of the TCP layer. This is particularly interesting in the TCP-UDP/IP based profile in the case, when the server does not have a public IP address. In this case, as the Client does not “see” the physical device hosting the server, it is not able to establish the required TCP layer connection.

In order to allow the server to establish the TCP layer connection, the server side COSEM TCP layer provides also TCP-CONNECT.request and.confirm services (and the Client side COSEM TCP layer provides also the TCP-CONNECT.indication and.response services).

NOTE These services are not used by the COSEM Application Layer, but directly by the Server (and Client) side TCP connection manager.

B.3.6.8 The COSEM TCP-UDP/IP profile and real-world IP networks

This standard and IEC 62056-47 specify all COSEM specific elements necessary to use COSEM over the Internet, using the COSEM TCP-UDP/IP based profile.

On real Internet networks, there are other elements, which need to be considered.

For example, in this standard it is specified, that physical devices hosting COSEM APs are identified with an IP address, but it is not specified, how to obtain such an IP address.

As these elements are not specific to COSEM, they are not in the scope of this standard.

Annex C
(informative)

AARQ and AARE encoding examples

C.1 General

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

In COSEM, the AARQ and AARE APDUs (see page The ACSE APDUs90) shall be encoded in BER (ISO/IEC 8825). In the user-information field, they contain the xDLMS Initiate.Request/.Response or DLMS ConfirmedServiceError PDUs respectively, encoded in A-XDR as OCTETSTRING.

C.2 Encoding the xDLMS-Initiate.request PDU

Firstly, the xDLMS-Initiate.request PDU. It is specified as follows:

```
xDLMS-Initiate.request:: = SEQUENCE
{
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,
client-max-receive-pdu-size          Unsigned16
}
```

where the proposed-conformance parameter holds the COSEM conformance block proposed by the client, as it is specified in 8.5.

Supposing that the client would like to build the AA with the following xDLMS context:

- no ciphering is used (the OPTIONAL dedicated-key is not present);
- response-allowed = TRUE (it holds the default value);
- no proposed-quality-of-service (the OPTIONAL proposed-quality-of-service parameter is not present);
- the proposed-dlms-version-number is 6 (xDLMS);
- the proposed-conformance – proposing all possible services and special features for both LN and SN referencing – is the following:

	Bit_00	Bit_01	Bit_02	Bit_03	Bit_04	Bit_05	Bit_06	Bit_07	Bit_08	Bit_09	Bit_10	Bit_11	Bit_12	Bit_13	Bit_14	Bit_15	Bit_16	Bit_17	Bit_18	Bit_19	Bit_20	Bit_21	Bit_22	Bit_23	Value of the BIT STRING
LN	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	00 7E 1F
SN	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1C 03 20

The meaning of this conformance block for LN referencing is:

- Attibute_0 referencing with SET is not supported (Bit_08)
- Priority Management is supported (Bit_09)
- Attibute_0 referencing with GET is supported (Bit_10)
- Block Transfer with the GET service is supported (Bit_11)

- Block Transfer with the SET service is supported (Bit_12)
- Block Transfer with the ACTION service is supported (Bit_13)
- Multiple references are supported (Bit_14)
- All LN services (GET, SET, ACTION, EVENT NOTIFICATION) are supported (Bit_19, 20, 22, 23)
- Selective Access feature is supported (Bit_21)

The meaning of this conformance block for SN referencing is:

- all SN services (READ, WRITE, UNCONFIRMED WRITE (information report)) are supported (Bit_03, 04, 05, 15)
- multiple references are supported (Bit_14)
- Parametrized_access is supported (Bit_18)

- The client-max-receive-pdu-size is 1200_D = 0x4B0.

With these parameters, the A-XDR encoding for the xDLMS-Initiate.request PDU:

```
-- A-XDR encoding the xDLMS-Initiate.request PDU
01      // encoding the tag (explicit tag) of the DLMS PDU CHOICE (InitiateRequest)
-- encoding the dedicated-key component (OPTIONAL, not present)
 00     // usage flag for the dedicated-key component (FALSE, not present)
-- encoding the response-allowed component (TRUE, default value)
  00    // usage flag for the response-allowed component (FALSE, default value conveyed)
-- encoding the proposed-quality-of-service component (OPTIONAL, not present)
   00   // usage flag for the proposed-quality-of-service component (FALSE, not present)
-- encoding the proposed-dlms-version-number component (Unsigned8, value=6)
    06  // the A-XDR encoding of an Unsigned8 is its value
-- encoding the Conformance block [APPLICATION 31] IMPLICIT BITSTRING (SIZE(24))
5F 1F   // encoding the [APPLICATION 31] tag (ASN.1 explicit tag)
 04     // encoding the length of the 'contents' field in octet (4)
  00    // encoding of the number of unused bits in the final octet of the BITSTRING
```

IMPORTANT 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.

LN referencing	SN referencing
00 7E 1F // encoding of the fixed length bitstring value	1C 03 20 // encoding of the fixed length bitstring value

```
-- encoding the client-max-receive-pdu-size component (Unsigned16, value=0x4B0)
04 B0   // the A-XDR encoding of an Unsigned16 is its value
```

Therefore, the A-XDR encoding of the xDLMS-Initiate.request PDU, with the given parameters, results in the following octet sequence:

LN referencing	SN referencing
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

This octet sequence shall be inserted into the user-information field of the AARQ APDU as an OCTET STRING.

C.3 Encoding the AARQ APDU not using the ACSE security mechanism

For ACSE use, suppose that the client would like to build the AA with the following application context:

- protocol-version is the default ACSE version;
- application-context-name:

LN referencing	SN referencing
{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) application-context(1) context_id(1)}	{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) application-context(1) context_id(2)}

- no authentication is used: neither the mechanism-name, nor the calling-authentication-value is present;
- no implementation-information is included.

The (BER) encoding of the AARQ APDU, corresponding to these parameters is as follows:

```
-- BER encoding the AARQ APDU
60       // encoding the tag for the AARQ APDU ([APPLICATION 0], Application)
 1D      // encoding of the length of the AARQ's content's field (29 octets)
-- no encoding for the protocol version, thus it is considered with its DEFAULT value

-- encoding the application-context-name component  (tagged component [1])
  A1     // encoding the tag for the application-context-name component ([1], Context-specific)
   09    // encoding of the length of the tagged component's value field

-- encoding the application-context-name component  (OBJECT IDENTIFIER)
    06   // encoding the choice for application-context-name  (OBJECT IDENTIFIER, Universal)
     07  // encoding of the length of the Object Identifier's value field (7 octets)
```

// encoding of the value of the Object Identifier[30]

LN referencing	SN referencing
60 85 74 05 08 01 01	60 85 74 05 08 01 02

```
-- encoding the user-information field component (tagged component, [30])
BE       // encoding the tag for the user-information field component ([30], Context-specific)
 10      // encoding of the length of the tagged component's value field.

-- encoding the user-information field component (OCTET STRING)
  04     // encoding the choice for user-information (OCTET STRING, Universal)
    0E   // encoding of the length of the OCTET STRING's value field (14 octets)

         // Here is the octet sequence of the xDLMS-Initiate.request PDU:
```

LN referencing	SN referencing
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

Therefore, the complete encoding for an AARQ APDU, with the given parameters is as follows (all values are in hexadecimals):

LN referencing	SN referencing
AARQ-pdu = [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]	AARQ-pdu = [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]

[30] BER Encoding for Object Identifier is a packed sequence of numbers representing the arc labels. Each number – except the first two, which are combined into one – is represented as a series of octets, with 7 bits being used from each octet and the most significant bit is set to 1 in all but the last octet. The fewest possible number of octets must be used.

For the case of the Object Identifer of this example (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*First+Second -> 40*2 + 16 = 96 = 0x60. The third number of the Object Identifier (756) requires two octets: its hexadecimal value is 0x2F4, which is 00000010 11110100, but following the above rule, the MSB of the second octet must be set to 0, thus this MSB should be shifted into the first octet, and to set the MSB of the first octet to 1, which gives binary 10000101 01110100, which is 0x8574. Each remaining number of the Object Identifier required to be encoded on one octet results in the above 60 85 74 05 08 01 01 encoding.

C.4 Encoding the AARQ APDU using low level authentication

The coding is as in the above example of Clause C.33, with the only difference of three additional fields that have to be encoded. These fields are:

- sender-acse-requirements: indicating that the ACSE functional unit is selected;
- mechanism-name: default-COSEM-low-level-security-mechanism-name **{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) authentication_mechanism_name(2) mechanism_id(1)};**
- calling-authentication-value: a GraphicString containing the password (assumed to be "12345678").

NOTE These three fields are coded immediately before the user-information-field.

--encoding the sender-acse-requirements field component (tagged component, [10])

8A // encoding the tag for the acse-requirements field component ([10], IMPLICIT), Context-specific

02 // encoding of the length of the tagged component's value field.

-- *encoding the sender-acse-requirements component (ACSE-requirements::= BIT STRING)*

07 // encoding the number of unused bits in the last byte of the BIT STRING

80 // encoding of the authentication functional unit (0)

NOTE The number of bits coded may vary from client to client, but within the COSEM environment, only bit 0 set to 1 (indicating the requirement of the authentication functional unit) is to be respected.

-- *encoding the mechanism-name component (tagged component [11])*

8B // encoding the tag for the mechanism-name component ([11], IMPLICIT), Context-specific

07 // encoding of the length of the tagged component's value field

-- *encoding of the value of the Object Identifier*

60 85 74 05 08 02 01

-- *encoding the calling-authentication-value component (tagged component [12])*

AC // encoding the tag for the calling-authentication-value component ([12], Context-specific)

0A // encoding of the length of the tagged component's value field

-- *encoding the calling-authentication-value component (Authentication-information::= CHOICE)*

80 // encoding the choice for Authentication-information (charstring [0] IMPLICIT GraphicString)

08 // encoding of the length of the Authentication-information's value field (8 octets)

-- *encoding of the value of the Password (GraphicString "12345678")*

31 32 33 34 35 36 37 38

Therefore, the complete encoding for an AARQ APDU, with the given parameters is as follows (all values are in hexadecimals):

LN referencing	SN referencing
AARQ-pdu = [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]	AARQ-pdu = [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]

C.5 Encoding the AARQ APDU using high-level authentication

The coding is as in the above example of Clause C.44, with the only difference in the content of the two fields:

- mechanism-name: default-COSEM-high-level-security-mechanism-name;
- calling-authentication-value: assumed no client-to-server challenge requested.

```
-- encoding the mechanism-name component (tagged component [11])
8B      // encoding the tag for the mechanism-name component ([11], IMPLICIT, Context-specific)
 07     // encoding of the length of the tagged component's value field
-- encoding of the value of the Object Identifier (default-COSEM-high-level-security-mechanism-name)
  60 85 74 05 08 02 02
-- encoding the calling-authentication-value component  (tagged component [12])
AC      // encoding the tag for the calling-authentication-value component ([12], Context-specific)
 02     // encoding of the length of the tagged component's value field
-- encoding the calling-authentication-value component  (Authentication-information::= CHOICE)
   80   // encoding the choice for Authentication-information (charstring [0] IMPLICIT GraphicString)
    00  // encoding of the length of the Authentication-information's value field (8 octets)
-- encoding of the value of the Password (GraphicString)
        // as the string is empty, no coding needed
```

Therefore, the complete encoding for an AARQ APDU, with the given parameters is as follows (all values are in hexadecimals):

LN referencing	SN referencing
AARQ-pdu = [60 2E A1 09 06 07 60 85 74 05 08 01 01 8A 02 07 80 8B 07 60 85 74 05 08 02 02 AC 02 80 00 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 00 7E 1F 04 B0]	AARQ-pdu = [60 2E A1 09 06 07 60 85 74 05 08 01 02 8A 02 07 80 8B 07 60 85 74 05 08 02 02 AC 02 80 00 BE 10 04 0E 01 00 00 00 06 5F 1F 04 00 1C 03 20 04 B0]

C.6 Encoding the AARE APDU, case of success

Recall the ASN.1 specification of the AARE APDU:

```
AARE-apdu::= [APPLICATION 1] IMPLICIT SEQUENCE
{
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] IMPLICIT Association-information OPTIONAL
}
```

In COSEM, this APDU shall be encoded in BER, and the user-information field contains an A-XDR encoded xDLMS-Initiate.response PDU as OCTETSTRING.

C.7 Encoding the xDLMS-Initiate.response PDU

The xDLMS-Initiate.response is specified as follows:

```
xDLMS-Initiate.response:: = SEQUENCE
{
        negotiated-quality-of-service        [0] IMPLICIT Integer8 OPTIONAL,
        negotiated-dlms-version-number       Unsigned8,
        negotiated-conformance               Conformance,
        server-max-receive-pdu-size          Unsigned16,
        vaa-name                             ObjectName
}
```

where the ObjectName type is specified as ObjectName::= Unsigned16, and the negotiated-conformance parameter contains the xDLMS services and features supported by the server.

Supposing that the server accepts the proposed AA within the following xDLMS context:

- no negotiated-quality-of-service (the OPTIONAL proposed-quality-of-service parameter is not present);
- the negotiated DLMS version number is 6 (xDLMS);
- the accepted xDLMS conformance block – which indicates the accepted services and special features for both LN and SN referencing – shall be as follows:

	Bit_00	Bit_01	Bit_02	Bit_03	Bit_04	Bit_05	Bit_06	Bit_07	Bit_08	Bit_09	Bit_10	Bit_11	Bit_12	Bit_13	Bit_14	Bit_15	Bit_16	Bit_17	Bit_18	Bit_19	Bit_20	Bit_21	Bit_22	Bit_23	Value of the BITSTRING
LN	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	1	1	1	1	00 50 1F
SN	0	0	0	1	1	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	1C 03 20

The meaning of this conformance block for LN referencing is:

Attibute_0 referencing with SET is not supported	(Bit_08)
Priority Management is supported	(Bit_09)
Attibute_0 referencing with GET is not supported	(Bit_10)
Block Transfer with the GET service is supported	(Bit_11)
Block Transfer with the SET service is not supported	(Bit_12)
Block Transfer with the ACTION service is not supported	(Bit_13)
Multiple references are not supported	(Bit_14)
All LN services (GET, SET, ACTION, EVENT NOTIFICATION) are supported	(Bit_19, 20, 22, 23)
Selective Access feature is supported	(Bit_21)

For SN referencing the meaning is as follows:

All SN services (READ, WRITE, UNCONFIRMED WRITE (Information Report)) are supported	(Bit_03, 04, 05, 15)
Multiple references are supported	(Bit_14)
Parametrized_access is supported	(Bit_18)

- the server-max-receive-pdu-size is 500_D = 0x1F4;
- vaa-name.

LN referencing	SN referencing
A dummy VAA name (0x0007) is assigned to this association.	The fixed Short Name of the standard association SN (0xFA00) is returned.

The A-XDR encoding of the xDLMS-Initiate.response PDU with these parameters is as follows:

```
-- A-XDR encoding the xDLMS-Initiate.response PDU
08       // encoding the tag (explicit tag) of the DLMS PDU CHOICE (InitiateResponse)
-- encoding the negotiated-quality-of-service component (OPTIONAL, not present)
 00      // usage flag for the negotiated-quality-of-service component (FALSE, not present)
-- encoding of the negotiated-dlms-version-number component (Unsigned8, value=6)
  06     // the A-XDR encoding of an Unsigned8 is its value
-- encoding the conformance block [APPLICATION 31] IMPLICIT BITSTRING (SIZE(24))
   5F 1F // encoding the [APPLICATION 31] tag (ASN.1 explicit tag)
    04   // encoding the length of the 'contents' field in octet (4)
     00  // encoding of the number of unused bits in the final octet of the bitstring

         // encoding of the fixed length BITSTRING value
```

LN referencing	SN referencing
00 50 1F	1C 03 20

```
-- encoding the server-max-receive-pdu-size component (Unsigned16, value=0x01F4)
01 F4    // the A-XDR encoding of an Unsigned16 is its value

-- encoding the VAA-Name component (Unsigned16, value=0x0007 for LN and FA 00 for SN)
         // the A-XDR encoding of an Unsigned16 is its value
```

LN referencing	SN referencing
00 07	FA 00

Thus, the A-XDR encoding of the xDLMS-Initiate.response PDU, with the above parameters, results in the following octet sequence:

LN referencing	SN referencing
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

This octet sequence shall be inserted into the user-information field of the AARE APDU as an OCTET STRING.

C.8 Encoding the AARE APDU not using security or using low level security

For ACSE use, supposing that the server accepts the proposed AA within the following application context:

- protocol-version is the default ACSE version;
- application-context-name:

LN referencing	SN referencing
{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) application-context(1) context_id(1)}	{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) application-context(1) context_id(2)}

- no additional authentication is used (low level authentication or without authentication): neither the mechanism-name, nor the calling-authentication-value are present;
- no implementation-information is included.

The (BER) encoding of the AARE PDU, corresponding to these parameters is as follows:

```
-- BER encoding the AARE APDU
61       // encoding the tag for the AARE-pdu ([APPLICATION 1], Application)
 29      // encoding of the length of the AARE’s content’s field (41 octets)
         // no encoding for the protocol version, thus it is considered with its DEFAULT value
```

```
-- encoding the application-context-name component (tagged component [1])
A1      // encoding the tag for the application-context-name component ([1], Context-specific )
 09     // encoding of the length of the tagged component's value field
-- encoding the application-context-name component  (OBJECT IDENTIFIER)
   06   // encoding the choice for application-context-name  (OBJECT IDENTIFIER, Universal)
    07  // encoding of the length of the Object Identifier's value field (7 octets)

// encoding of the value of the Object Identifier
```

LN referencing	SN referencing
60 85 74 05 08 01 01	60 85 74 05 08 01 02

```
-- encoding the tag & length for the result component (tagged component [2] )
A2      // encoding the tag & length for the result component ([2], Context-specific )
 03     // encoding of the length of the tagged component's value field

-- encoding the result-component  (INTEGER)
  02    // encoding the choice for result  (INTEGER, Universal)
   01   // encoding of the length of the result's value field (1 octets)
    00  // encoding of the value of the Result (0, accepted)

-- encoding the result-source-diagnostic (tagged component [3])
A3      // encoding the tag for the result-source-diagnostic component ([3], Context-specific )
 05     // encoding of the length of the tagged component's value field
  A1    // encoding the tag for the acse-service-user CHOICE (1)
   03   // encoding of the length of the tagged component's value field
-- encoding the result-source-diagnostics component  (INTEGER)
    02  // encoding the choice for result-source-diagnostics  (INTEGER, Universal)
     01 // encoding of the length of the value field (1 octets)
      00 // encoding of the value: 0, no diagnostics provided.

-- encoding the user-information field component (tagged component, [30])
BE      // encoding the tag for the user-information field component ([30], Context-specific )
 10     // encoding of the length of the tagged component's value field
        -- encoding the user-information field component (OCTET STRING)
  04    // encoding the choice for user-information  (OCTET STRING, Universal)
   0E   // encoding of the length of the OCTET STRING's value field (14 octets )

// Here is the octet sequence of the xDLMS-Initiate.response PDU
```

LN referencing	SN referencing
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

Therefore, the complete encoding for an AARE APDU, with the given parameters is as follows (all values are in hexadecimals):

LN referencing	SN referencing
AARE-pdu = [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]	AARE-pdu = [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]

C.9 Encoding the AARE APDU using high level security

The coding is as in the above example of Clause C.88, with the following differences:

a) three additional fields that have to be coded:

- responder-acse-requirements: indicating that the ACSE functional unit is selected;
- mechanism-name: default-COSEM-high-level-security-mechanism-name **{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) authentication_mechanism_name(2) mechanism_id(2)};**
- responding-authentication-value: A GraphicString containing the server-to-client challenge (assumed to be “P6wRJ21”);

b) in result-source-diagnostic the value is set to “authentication-required” to reflect the additional authentication step needed.

NOTE The three additional fields are coded immediately before the user-information-field.

```
-- encoding the responder-acse-requirements field component (tagged component, [8])
88      // encoding the tag for the acse-requirements field component ([8], IMPLICIT,
          Context-specific)
 02     // encoding of the length of the tagged component’s value field.
-- encoding the responder-acse-requirements component (ACSE-requirements::= BIT
STRING)
  07    // encoding the number of unused bits in the last byte of the BIT STRING
   80   // encoding of the authentication functional unit (0)
-- encoding the mechanism-name component (tagged component [9])
89      // encoding the tag for the mechanism-name component ([9], IMPLICIT,
          Context-specific)
 07     // encoding of the length of the tagged component’s value field
-- encoding of the value of the Object Identifier (default-COSEM-high-level-security-
mechanism-name)
  60 85 74 05 08 02 02
-- encoding the responding-authentication-value component (tagged component [10])
AA      // encoding the tag for the responding-authentication-value component ([10], Context-
specific)
 0A     // encoding of the length of the tagged component’s value field
-- encoding the responding-authentication-value component (Authentication-information::=
CHOICE)
  80    // encoding the choice for Authentication-information (charstring [0] IMPLICIT
GraphicString)
  08    // encoding of the length of the Authentication-information’s value field (8 octets)
-- encoding of the value of the Password (GraphicString)
   50 36 77 52 4A 32 31
```

The result-source-diagnostic is coded as follows:

```
-- encoding the result-source-diagnostic (tagged component [3])
A3      // encoding the tag for the result-source-diagnostics component ([3], Context-specific )
 05     // encoding of the length of the tagged component’s value field
  A1    // encoding the tag for the acse-service-user CHOICE (1)
  03    // encoding of the length of the tagged component’s value field
-- encoding the result-source-diagnostics component (INTEGER)
   02   // encoding the choice for result-source-diagnostics (INTEGER, Universal)
    01  // encoding of the length of the value field (1 octets)
     0E // encoding of the value: 14, authentication-required
```

Therefore, the complete encoding for an AARE APDU, with the given parameters is as follows (all values are in hexadecimals):

LN referencing	SN referencing
AARE-pdu = [61 41 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 02 AA 0A 80 08 50 36 77 52 4A 32 31 BE 10 04 0E 08 00 06 5F 1F 04 00 00 50 1F 01 F4 00 07]	AARE-pdu = [61 41 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 02 AA 0A 80 08 50 36 77 52 4A 32 31 BE 10 04 0E 08 00 06 5F 1F 04 00 1C 03 20 01 F4 FA 00]

C.10 Encoding the AARE APDU case of failure 1

This example shows the construction of an AARE-pdu, when the server is not able to accept the proposed association because of the received application-context-name does not fit to the application context, which can be supported by the server.

In this case, the ‘result’ field of the AARE PDU shall contain the ‘rejected-permanent’ value, the ‘result-source-diagnostic’ field the ‘application-context-name-not-supported’ value, and – supposing that the server is able to support the proposed xDLMS context – the user-information field shall contain a correctly constructed (encoded en A-XDR, as a BER OCTET STRING) xDLMS-Initiate.response PDU. (It is the same as that of the previous example).

Thus, the A-XDR encoding of the xDLMS-Initiate.response PDU shall be as follows:

LN referencing	SN referencing
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

Parameters for the AARE:

- protocol-version is the default ACSE version;
- application-context-name: COSEM_Application_Context_Name-Logical_Name_Referencing (the proposed) **{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) application-context(1) context_id(1)};**
- no authentication is used: neither the mechanism-name, nor the calling-authentication-value are present;
- no implementation-information is included;
- result = rejected-permanent;
- associate-source-diagnostics = application-context-name-not-supported.

The (BER) encoding of the AARE APDU, corresponding to these parameters is as follows:

```
-- BER encoding the AARE APDU
61       // encoding the tag for the AARE-pdu ([APPLICATION 1], Application)
 29      // encoding of the length of the AARE’s content’s field (41 octets)
         // no encoding for the protocol version, thus it is considered with its DEFAULT value
-- encoding the application-context-name component  (tagged component [1])
  A1     // encoding the tag for the application-context-name component ([1], Context-specific )
   09    // encoding of the length of the tagged component’s value field
-- encoding the application-context-name component  (OBJECT IDENTIFIER)
    06   // encoding the choice for application-context-name  (OBJECT IDENTIFIER, Universal)
     07  // encoding of the length of the Object Identifier’s value field (7 octets)
```

LN referencing	SN referencing
60 85 74 05 08 01 01 // encoding of the value of the Object Identifier (2,16,756,5,8,1,1)	60 85 74 05 08 01 02 // encoding of the value of the Object Identifier (2,16,756,5,8,1,2)

```
-- encoding the tag & length for the result component  (tagged component [2])
A2       // encoding the tag for the result component ([2], Context-specific )
 03      // encoding of the length of the tagged component’s value field
-- encoding the Result component (INTEGER)
  02     // encoding the choice for result (INTEGER, Universal)
   01    // encoding of the length of the result’s value field (1 octets)
    01// encoding of the value of the result (1, rejected-permanent)
-- encoding the result-source-diagnostics component (tagged component [3])
A3       // encoding the tag for the result-source-diagnostics component ([3], Context-specific )
 05      // encoding of the length of the tagged component’s value field
  A1     // encoding the tag for the acse-service-user CHOICE (1)
   03    // encoding of the length of the tagged component’s value field
-- encoding the result-source-diagnostics component  (INTEGER)
    02   // encoding the choice for result-source-diagnostics  (INTEGER, Universal)
     01  // encoding of the length of the value field (1 octets)
      02 // encoding of the value: 2, application-context-name-not-supported
-- encoding the user-information field component (tagged component, [30])
BE       // encoding the tag for the user-information field component ([30], Context-specific )
 10      // encoding of the length of the tagged component’s value field
-- encoding the user-information component  (OCTET STRING)
  04     // encoding the choice for user-information  (OCTET STRING, Universal)
   0E    // encoding of the length of the OCTET STRING’s value field (14 octets )
```

// Here is the octet sequence of the xDLMS-Initiate.response PDU:

LN referencing	SN referencing
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

Therefore, the complete encoding for an AARE APDU with the given parameters is as follows (all values are in hexadecimals):

LN referencing	SN referencing
AARE-pdu = [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]	AARE-pdu = [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 1C 03 20 01 F4 FA 00]

C.11 Encoding the AARE APDU, case of failure 2

This example shows the construction of an AARE APDU, when the server is not able to accept the proposed AA because the proposed xDLMS context cannot be supported by the server (for the reason that “the proposed DLMS version number is too low”). The proposed COSEM application context could be accepted.

In this case, the ‘result’ field of the AARE APDU shall contain the ‘rejected-permanent’ value, the ‘result-source-diagnostic’ field the ‘no-reason-given’ value, and the user-information field shall contain a correctly constructed (encoded en A-XDR, as a BER OCTET STRING) DLMS-ConfirmedServiceError PDU, indicating the reason for the failure.

The DLMS ConfirmedServiceError message is specified as follows:

```
ConfirmedServiceError::=        CHOICE
{
        -- tag 0 is reserved
        initiateError           [1]     ServiceError,
        getStatus               [2]     ServiceError,
        getNameList             [3]     ServiceError,
        terminateUpLoad         [19]    ServiceError
}
```

where ServiceError is as follows:

```
ServiceError    ::=     CHOICE
{
        initiate                [6]     IMPLICIT ENUMERATED
        -- initiate service error
        {
                other                           (0),
                DLMS-version-too-low            (1), -- proposed DLMS version too low
                incompatible-conformance        (2), -- proposed services not sufficient
                PDU-size-too-short              (3), -- proposed PDU size too short
                refused-by-the-VDE-Handler      (4)   -- vaa creation impossible or not allowed
        }
}
```

Therefore, A-XDR encoding of the DLMS ConfirmedServiceError PDU with the above conditions is as follows:

```
-- A-XDR encoding the DLMS ConfirmedServiceError-pdu
0E      // encoding the tag  (explicit tag) of the DLMS PDU CHOICE (ConfirmedServiceError)
-- encoding the tag for selecting InitiateError (InitiateError = 1)
 01     // tag (explicit) of ConfirmedServiceError CHOICE (InitiateError =1)
-- encoding the tag for selecting ServiceError (Initiate = 6)
  06    // tag (explicit) of ServiceError CHOICE (Initiate 6)
-- encoding the enumerated reason of failure (proposed DLMS version too low = 1)
   01   // encoding the value of the ENUMERATED type
```

Thus, the A-XDR encoding of the DLMS ConfirmedServiceError PDU, with the above parameters, results in the following octet sequence:

DLMS ConfirmedServiceError PDU: 0E 01 06 01 (for both LN and SN referencing)

This octet sequence shall be inserted into the user-information field of the AARE APDU as an OCTET STRING.

Supposing, that AARE parameters are as follows:

- protocol-version is the default ACSE version;
- application-context-name: COSEM_Application_Context_Name_Logical_Name_Referencing (the proposed) **{joint-iso-ccitt(2) country(16) country-name(756) identified-organization(5) DLMS-UA(8) application-context(1) context_id(1)};**
- no authentication is used: neither the mechanism-name, nor the calling-authentication-value is present;
- no implementation-information is included;
- result = rejected-permanent;
- associate-source-diagnostics = no-reason-given.

The (BER) encoding of the AARE-pdu, corresponding to these parameters is as follows:

```
-- BER encoding the AARE APDU
61        // encoding the tag for the AARE-pdu ([APPLICATION 1], Application)
 1F       // encoding of the length of the AARE’s content’s field (31 octets)
          // no encoding for the protocol version, thus it is considered with its DEFAULT value
-- encoding the application-context-name component  (tagged component [1])
A1        // encoding the tag for the application-context-name component ([1], Context-specific)
 09       // encoding of the length of the tagged component’s value field
-- encoding the application-context-name component  (OBJECT IDENTIFIER)
  06      // encoding the choice for application-context-name  (OBJECT IDENTIFIER, Universal)
   07     // encoding of the length of the Object Identifier’s value field (7 octets)
    60 85 74 05 08 01 01 // encoding of the value of the Object Identifier (2,16,756,5,8,1,1)
-- encoding the tag & length for the result component  (tagged component [2])
A2        // encoding the tag for the result component ([2], Context-specific)
 03       // encoding of the length of the tagged component’s value field
-- encoding the Result component  (INTEGER)
  02      // encoding the choice for result  (INTEGER, Universal)
   01     // encoding of the length of the result’s value field (1 octets)
    01    // encoding of the value of the result (1, rejected-permanent)
-- encoding the result-source-diagnostics component  (tagged component [3])
A3        // encoding the tag for the result-source-diagnostics component ([3], Context-specific )
 05       // encoding of the length of the tagged component’s value field
  A1      // encoding the tag for the acse-service-user CHOICE (1)
   03     // encoding of the length of the tagged component’s value field
-- encoding the result-source-diagnostics component  (INTEGER)
    02    // encoding the choice for result-source-diagnostics  (INTEGER, Universal)
     01   // encoding of the length of the value field (1 octets)
      01// encoding of the value: 1, no-reason-given

-- encoding the user-information field component (tagged component, [30])
BE        // encoding the tag for the user-information field component ([30], Context-specific)
 06       // encoding of the length of the tagged component’s value field
-- encoding the user-information component  (OCTET STRING)
  04      // encoding the choice for user-information  (OCTET STRING, Universal)
   04     // encoding of the length of the OCTET STRING’s value field
    0E 01 06 01   // Here is the octet sequence of the DLMS-ConfirmedServiceError-pdu
```

Therefore, the complete encoding for an AARE APDU using LN referencing, with the given parameters is as follows (all values are in hexadecimals):

LN referencing	SN referencing
AARE-pdu = [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]	AARE-pdu = [61 1F A1 09 06 07 60 85 74 05 08 01 02 A2 03 02 01 01 A3 05 A1 03 02 01 01 BE 06 04 04 0E 01 06 01]

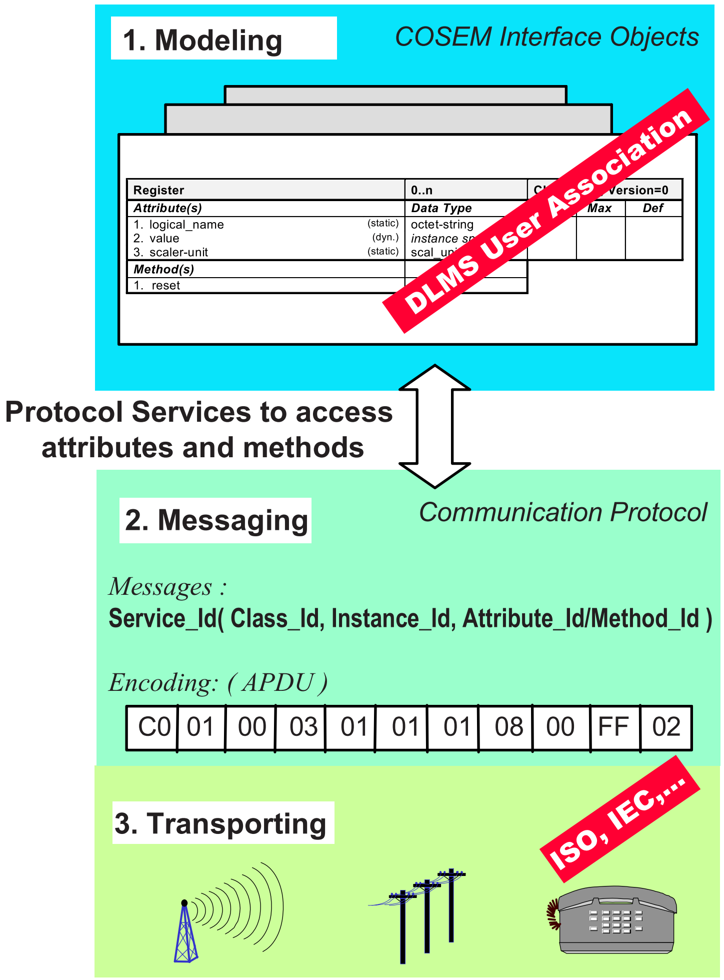
Annex D
(informative)

Data model and protocol

The data model uses generic building blocks to define the complex functionality of the metering equipment. It provides a view of this functionality of the meter, as it is available at its interface(s). The model does not cover internal, implementation specific issues.

The communication protocol defines how the data can be accessed and exchanged.

This is illustrated in the figure below:



IEC 2109/06

Figure D.1 – The three-step approach of COSEM

- the COSEM specification specifies metering domain specific interface classes. The functionality of the meter is defined by the instances of these interface classes, called COSEM interface objects. This is defined in IEC 62056-62. Logical names, identifying the COSEM interface objects are defined in IEC 62056-61;
- the attributes and methods of these COSEM interface objects can be accessed and used via the messaging services of the application layer;
- the lower layers of the protocol transport the information.

Bibliography

IEC 62056-41:1998, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 41: Data exchange using wide area networks: Public switched telephone network (PSTN) with LINK+ protocol*

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

IEC 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*

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

NOTE Harmonized as EN ISO/IEC 7498-1:1995 (not modified).

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*

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

RFC 0768 – *User Datagram Protocol*
Author: J. Postel
Date: Aug-28-1980
Also: STD0006

RFC 0791 – *Internet Protocol*
Author: J. Postel
Date: Sep-01-1981
Also: STD0005
Updated by: RFC1349
Obsoletes: RFC0760

RFC 0792 – *Internet Control Message Protocol*
Author: J. Postel
Date: Sep-01-1981
Also: STD0005
Updated by: RFC0950
Obsoletes: RFC0777

RFC 0793 – *Transmission Control Protocol*
Author: J. Postel
Date: Sep-01-1981
Also: STD0007
Updated by: RFC3168

RFC 0826 – *Ethernet Address Resolution Protocol: Or converting network protocol addresses to 48.bit Ethernet address for transmission on Ethernet hardware*
Author: D.C. Plummer
Date: Nov-01-1982
Also: STD0037

RFC 0894 – *Standard for the transmission of IP datagrams over Ethernet networks*
Author: C. Hornig
Date: Apr-01-1984
Also: STD0041

RFC 0919 – *Broadcasting Internet Datagrams*
Author: J.C. Mogul
Date: Oct-01-1984
Also: STD0005

RFC 0922 – *Broadcasting Internet datagrams in the presence of subnets*
Author: J.C. Mogul
Date: Oct-01-1984
Also: STD0005

RFC 0950 – *Internet Standard Subnetting Procedure*
Authors: J.C. Mogul, J. Postel
Date: Aug-01-1985
Also: STD0005
Updates: RFC0792

RFC 1042 – *Standard for the transmission of IP datagrams over IEEE 802 networks*
Authors: J. Postel, J.K. Reynolds
Date: Feb-01-1988
Also: STD0043
Obsoletes: RFC0948

RFC 1112 – *Host extensions for IP multicasting*
Author: S.E. Deering
Date: Aug-01-1989
Also: STD0005
Updated by: RFC2236
Obsoletes: RFC0988, RFC1054

RFC 1321 – The MD5 Message-Digest Algorithm
Author: R. Rivest
Date: April 1992

RFC 1332 – *The PPP Internet Protocol Control Protocol (IPCP)*
Author: G. McGregor
Date: May 1992
Updated by: RFC3241
Obsoletes: RFC1172

RFC 1661 – *The Point-to-Point Protocol (PPP)*
Authors: W. Simpson, Ed.
Date: July 1994
Also: STD0051
Updated by: RFC2153
Obsoletes: RFC1548

RFC 1662 – *PPP in HDLC-like Framing*
Authors: W. Simpson, Ed.
Date: July 1994
Also: STD0051
Obsoletes: RFC1549

RFC 1755 – *ATM Signaling Support for IP over ATM*
Authors: M. Perez, F. Liaw, A. Mankin, E. Hoffman, D. Grossman, A. Malis
Date: February 1995

RFC 1962 – *The PPP Compression Control Protocol (CCP)*
Author: D. Rand
Date: June 1996
Updated by: RFC2153

RFC 2131 – *Dynamic Host Configuration Protocol*
Author: R. Droms
Date: March 1997
Updated by: RFC3396
Obsoletes: RFC1541

RFC 2153 – *PPP Vendor Extensions*
Author: W. Simpson
Date: May 1997
Updates: RFC1661, RFC1962

RFC 2225 – *Classical IP and ARP over ATM*
Authors: M. Laubach, J. Halpern
Date: April 1998
Obsoletes: RFC1626, RFC1577

FIPS PUB 180-1 SECURE HASH STANDARD
Date: 1993 May 11
Supersedes FIPS PUB 180

INDEX

Annex ZA
(normative)

Normative references to international publications with their corresponding European publications

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

NOTE When an international publication has been modified by common modifications, indicated by (mod), the relevant EN/HD applies.

Publication	Year	Title	EN/HD	Year
IEC 60050-300	2001	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 61334-4-41	1996	Distribution automation using distribution line carrier systems - Part 4: Data communication protocols - Section 41: Application protocols - Distribution line message specification	EN 61334-4-41	1996
IEC 61334-6	2000	Distribution automation using distribution line carrier systems - Part 6: A-XDR encoding rule	EN 61334-6	2000
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	-	-
IEC 62056-21	2002	Electricity metering - Data exchange for meter reading, tariff and load control - Part 21: Direct local data exchange	EN 62056-21	2002
IEC 62056-42	2002	Electricity metering - Data exchange for meter reading, tariff and load control - Part 42: Physical layer services and procedures for connection-oriented asynchronous data exchange	EN 62056-42	2002
IEC 62056-46 A1	2002 2006	Electricity metering - Data exchange for meter reading, tariff and load control - Part 46: Data link layer using HDLC protocol	EN 62056-46 A1	2002 2007

Publication	Year	Title	EN/HD	Year
IEC 62056-47	2006	Electricity metering - Data exchange for meter reading, tariff and load control - Part 47: COSEM transport layers for IPv4 networks	EN 62056-47	2007
IEC 62056-61	2006	Electricity metering - Data exchange for meter reading, tariff and load control - Part 61: Object identification system (OBIS)	EN 62056-61	2007
IEC 62056-62	2006	Electricity metering - Data exchange for meter reading, tariff and load control - Part 62: Interface classes	EN 62056-62	2007
ISO/IEC 8649	1996	Information technology - Open systems interconnection - Service definition for the Association Control Service Element	-	-
ISO/IEC 8650-1	1996	Information technology - Open systems interconnection - Connection-oriented protocol for the association control service Information technology - ASN.1 encoding rules: Protocol specification	-	-
ISO/IEC 8824	Series	Information technology - Abstract Syntax Notation One (ASN.1)	-	-
ISO/IEC 8825	Series	Information technology - ASN.1 encoding rules	-	-
ISO/IEC 13239	2002	Information technology - Telecommunications and information exchange between systems - High-level data link control (HDLC) procedures	-	-
STD 0005	1981	Internet Protocol	-	-
STD 0006	1980	User Datagram Protocol	-	-
STD 0007	1981	Transmission Control Protocol	-	-

blank

BSI — British Standards Institution

BSI is the independent national body responsible for preparing British Standards. It presents the UK view on standards in Europe and at the international level. It is incorporated by Royal Charter.

Revisions

British Standards are updated by amendment or revision. Users of British Standards should make sure that they possess the latest amendments or editions.

It is the constant aim of BSI to improve the quality of our products and services. We would be grateful if anyone finding an inaccuracy or ambiguity while using this British Standard would inform the Secretary of the technical committee responsible, the identity of which can be found on the inside front cover. Tel: +44 (0)20 8996 9000. Fax: +44 (0)20 8996 7400.

BSI offers members an individual updating service called PLUS which ensures that subscribers automatically receive the latest editions of standards.

Buying standards

Orders for all BSI, international and foreign standards publications should be addressed to Customer Services. Tel: +44 (0)20 8996 9001. Fax: +44 (0)20 8996 7001. Email: orders@bsi-global.com. Standards are also available from the BSI website at http://www.bsi-global.com.

In response to orders for international standards, it is BSI policy to supply the BSI implementation of those that have been published as British Standards, unless otherwise requested.

Information on standards

BSI provides a wide range of information on national, European and international standards through its Library and its Technical Help to Exporters Service. Various BSI electronic information services are also available which give details on all its products and services. Contact the Information Centre. Tel: +44 (0)20 8996 7111. Fax: +44 (0)20 8996 7048. Email: info@bsi-global.com.

Subscribing members of BSI are kept up to date with standards developments and receive substantial discounts on the purchase price of standards. For details of these and other benefits contact Membership Administration. Tel: +44 (0)20 8996 7002. Fax: +44 (0)20 8996 7001. Email: membership@bsi-global.com.

Information regarding online access to British Standards via British Standards Online can be found at http://www.bsi-global.com/bsonline.

Further information about BSI is available on the BSI website at http://www.bsi-global.com.

Copyright

Details and advice can be obtained from the Copyright & Licensing Manager. Tel: +44 (0)20 8996 7070. Fax: +44 (0)20 8996 7553. Email: copyright@bsi-global.com.

BSI
389 Chiswick High Road
London
W4 4AL