INTERNATIONAL STANDARD

IEC 62056-53

First edition
2002-02

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

Part 53: COSEM application layer

Reference number
IEC 62056-53:2002(E)

Publication numbering

As from 1 January 1997 all IEC publications are issued with a designation in the 60000 series. For example, IEC 34-1 is now referred to as IEC 60034-1.

Consolidated editions

The IEC is now publishing consolidated versions of its publications. For example, edition numbers 1.0, 1.1 and 1.2 refer, respectively, to the base publication, the base publication incorporating amendment 1 and the base publication incorporating amendments 1 and 2.

Further information on IEC publications

The technical content of IEC publications is kept under constant review by the IEC, thus ensuring that the content reflects current technology. Information relating to this publication, including its validity, is available in the IEC Catalogue of publications (see below) in addition to new editions, amendments and corrigenda. Information on the subjects under consideration and work in progress undertaken by the technical committee which has prepared this publication, as well as the list of publications issued, is also available from the following:

- **IEC Web Site (www.iec.ch)**
- **Catalogue of IEC publications**

  The on-line catalogue on the IEC web site (www.iec.ch/catlg-e.htm) enables you to search by a variety of criteria including text searches, technical committees and date of publication. On-line information is also available on recently issued publications, withdrawn and replaced publications, as well as corrigenda.
- **IEC Just Published**

  This summary of recently issued publications (www.iec.ch/JP.htm) is also available by email. Please contact the Customer Service Centre (see below) for further information.
- **Customer Service Centre**

  If you have any questions regarding this publication or need further assistance, please contact the Customer Service Centre:

  Email: custserv@iec.ch
  Tel: +41 22 919 02 11
  Fax: +41 22 919 03 00

INTERNATIONAL STANDARD

IEC 62056-53

First edition
2002-02

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

Part 53: COSEM application layer

International Electrotechnical Commission 3, rue de Varembé Geneva, Switzerland
Telefax: +41 22 919 0300 e-mail: inmail@iec.ch IEC web site http://www.iec.ch

Commission Electrotechnique Internationale
International Electrotechnical Commission
Международная Электротехническая Комиссия

PRICE CODE XF

For price, see current catalogue

CONTENTS

INTERNATIONAL ELECTROTECHNICAL COMMISSION

____

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

Part 53: COSEM application layer

FOREWORD

1) The IEC (International Electrotechnical Commission) is a worldwide organization for standardization comprising all national electrotechnical committees (IEC National Committees). The object of the IEC is to promote international co-operation on all questions concerning standardization in the electrical and electronic fields. To this end and in addition to other activities, the IEC publishes International Standards. Their preparation is entrusted to technical committees; any IEC National Committee interested in the subject dealt with may participate in this preparatory work. International, governmental and non-governmental organizations liaising with the IEC also participate in this preparation. The IEC collaborates closely with the International Organization for Standardization (ISO) in accordance with conditions determined by agreement between the two organizations.
2) The formal decisions or agreements of the IEC on technical matters express, as nearly as possible, an international consensus of opinion on the relevant subjects since each technical committee has representation from all interested National Committees.
3) The documents produced have the form of recommendations for international use and are published in the form of standards, technical specifications, technical reports or guides and they are accepted by the National Committees in that sense.
4) In order to promote international unification, IEC National Committees undertake to apply IEC International Standards transparently to the maximum extent possible in their national and regional standards. Any divergence between the IEC Standard and the corresponding national or regional standard shall be clearly indicated in the latter.
5) The IEC provides no marking procedure to indicate its approval and cannot be rendered responsible for any equipment declared to be in conformity with one of its standards.

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

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

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

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

International Standard IEC 62056-53 has been prepared by IEC technical committee 13: Equipment for electrical energy measurement and load control.

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

FDIS	Report on voting
13/1268/FDIS	13/1274/RVD

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

[1] Device Language Message Specification.

This publication has been drafted in accordance with the ISO/IEC Directives, Part 3.

Annexes A and B form an integral part of this standard.

Annexes C and D are for information only.

The committee has decided that the contents of this publication will remain unchanged until 2006. At this date, the publication will be

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

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.

Data communication services with COSEM interface objects, using Logical name (LN) referencing and Short name (SN) referencing, are specified. COSEM servers use either LN or SN referencing during a given association: this is negotiated during the Application Association establishment. The COSEM client always uses LN referencing. If the client communicates with a server using SN referencing, the LN services are mapped to SN services.

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 – 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/TR2 62051:1999, *Electricity metering – Glossary of terms*

IEC 62056-21, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 21: Direct local data exchange* [2]

IEC 62056-42:2001, *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, *Electricity metering – Data exchange for meter reading, tariff and load control – Part 46: Data link layer using HDLC protocol*

[2] To be published.

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

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

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

ISO/IEC/TR2 8650-1:1996, *Information technology – Open systems interconnection – Connection-oriented protocol for the association control service element: Protocol specification*

ISO/IEC 8824:1990, *Information technology – Open Systems Interconnection – Specification of Abstract Syntax Notation One (ASN.1)*

ISO/IEC 8825:1990, *Information technology – Open Systems Interconnection – Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1)*

ISO/IEC 13239:2000, *Information technology – Telecommunications and information exchange between systems – High-level data link control (HDLC) procedures*

3 Terms, definitions and abbreviations

3.1 Terms and definitions

For the purpose of this part of IEC 62056, the definitions in IEC 60050-300 and IEC/TR 62051, as well as the following, apply.

3.1.1
base_name
the short_name corresponding to the first attribute (“logical_name”) of a COSEM interface object

3.1.2
class_id
interface class identification code

3.1.3
client
a station, asking for services

3.1.4
COSEM interface object
an instance of a COSEM interface class

3.1.5
server
a station, delivering services. The tariff device (metering equipment) is normally the server, delivering the requested data or executing the requested tasks.

3.2 Abbreviations

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
ASE	Application Service Element
ASO	Application Service Object
A-XDR	Adapted eXtended Data Representation
BER	Basic Encoding Rules
CF	Control function
.cnf	confirm service primitive
CO	Connection Oriented
COSEM	COmpanion Specification for Energy Metering
DLMS	Distribution Line Message Specification
DSAP	Data link Service Access Point
GMT	Greenwich Mean Time
HDLC	High-level Data Link Control
HLS	High-Level Security
IC	Interface Class
LLC	Logical Link Control (sub-layer)
LLS	Low Level Security
LPDU	LLC Protocol Data Unit
LSB	Least Significant Bit
LSAP	LLC sub-layer Service Access Point
m	mandatory, used in conjunction with attribute and method definitions
MSB	Most Significant Bit
MSC	Message Sequence Chart
o	optional, used in conjunction with attribute and method definitions
OBIS	OBject Identification System
PDU	Protocol Data Unit
.req	.request service primitive
.res	.response service primitive
SAP	Service Access Point
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 classes 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 application process: in other words, the server application process provides remote services to the client application process. These services are provided via exchanging messages (SERVICE.requests/.responses) between the client and the server application processes, as it is shown in Figure 1.

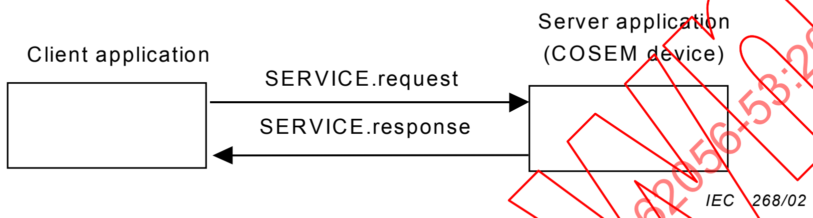


Figure 1 – Client/server relationship in COSEM

In general, the client and the server application processes are located in separate devices, exchanging messages is done with the help of the communications protocol.

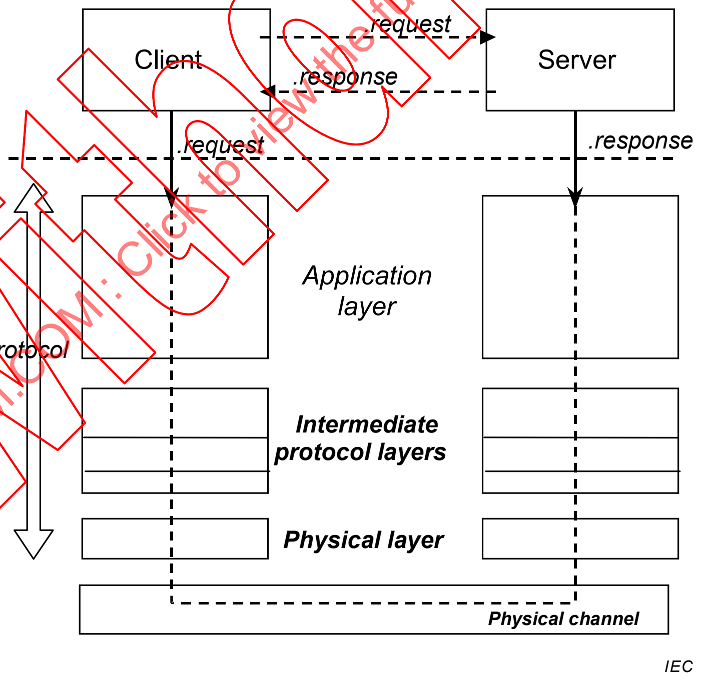


Figure 2 – Exchanging messages via the communications protocol

[3] The 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.

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 which shall contain 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 from the COSEM model, consequently the COSEM application layer can be placed on the top of a wide variety of lower protocol layer stacks, as it is shown in Figure 3.

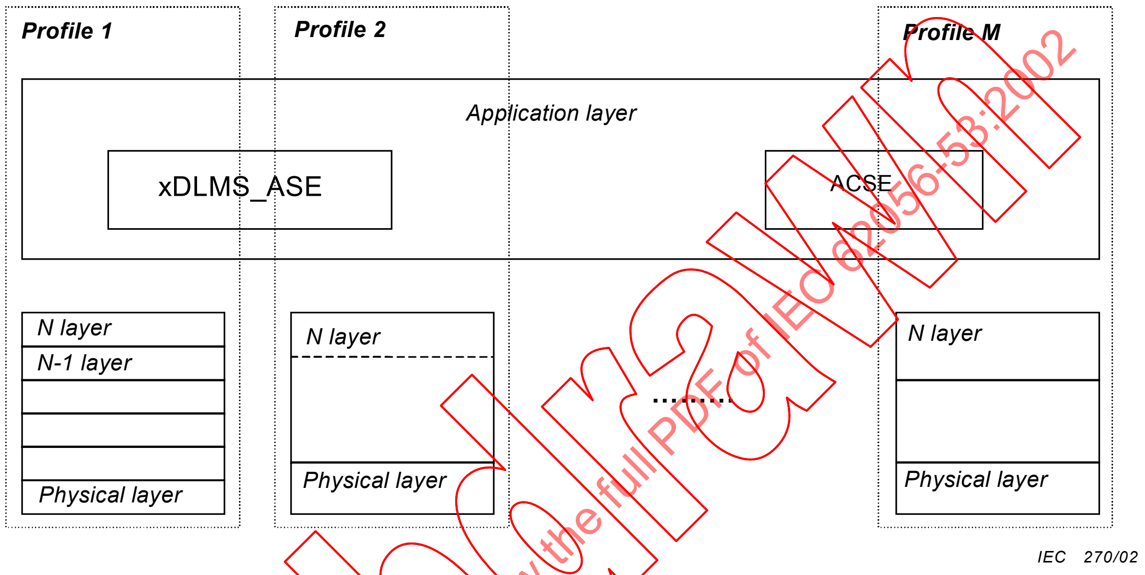


Figure 3 – The COSEM application layer on the top of various lower layer stacks

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

A communications 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 application processes can use the services of the xDLMS_ASE only when these application processes are associated[5]. Therefore, in this environment a communication session consists of three phases, as it is shown on Figure 4.

[4] ACSE = **A**ssociation **C**ontrol Service **E**lement

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

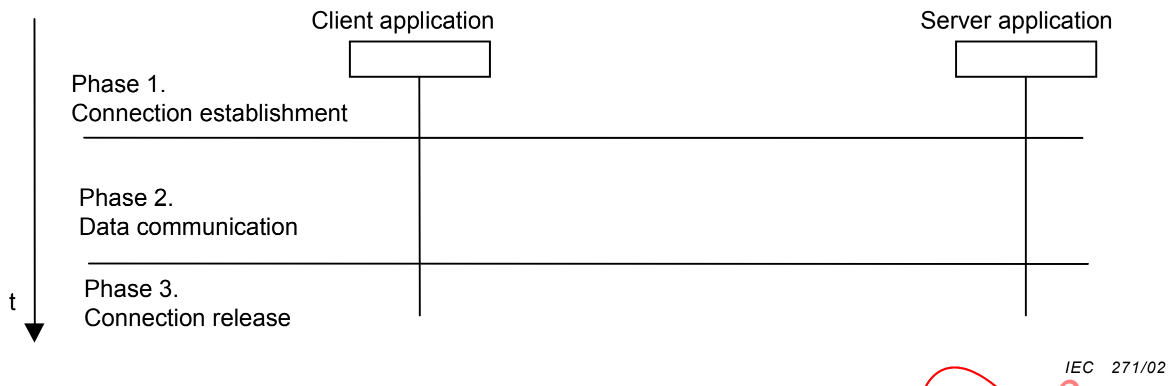


Figure 4 – A complete communications session in the CO environment

In the 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 6.3.2. For these 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 application process, and uses services provided by the supporting lower layer.

Both the client and server side COSEM ASO contains 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/TR2 8650-1 is used;
- the Extended DLMS application service element (xDLMS_ASE). The task of this element is to provide data communication services between COSEM equipment. See also Annex B;
- 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.

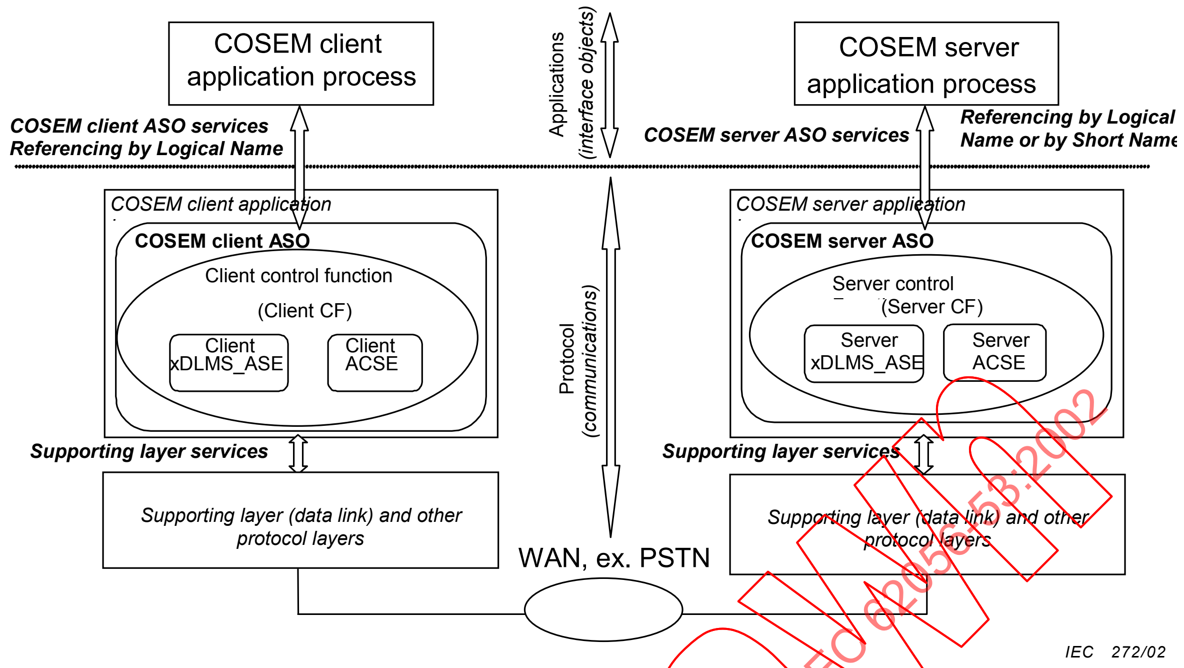

COSEM client application process
COSEM client ASO services
Referencing by Logical Name
Applications (interface objects)
COSEM server application process
COSEM server ASO services
Referencing by Logical Name or by Short Name
COSEM client application
COSEM client ASO
Client control function
(Client CF)
Client xDLMS_ASE
Client ACSE
Protocol (communications)
COSEM server application
COSEM server ASO
Server control (Server CF)
Server xDLMS_ASE
Server ACSE
Supporting layer services
Supporting layer services
Supporting layer (data link) and other protocol layers
WAN, ex. PSTN
Supporting layer (data link) and other protocol layers
IEC 272/02

Figure 5 – The structure of the COSEM application layers

5.3 Service specification

Service specifications cover the services required of, or by the COSEM client and server application processes 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 application association establishment phase and relies on the association request/response services of the ACSE. In the case of pre-established application associations (6.3.2) these services are not used.

As in any COSEM communications profile, there is a one-to-one relationship between an application association and a supporting protocol layer connection, the COSEM-RELEASE and COSEM-ABORT services – used during the connection release phase – do not rely on the ACSE. Application associations are released or aborted simply by disconnecting the corresponding supporting layer connection.

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), Write (SN);
- the EventNotification (LN), InformationReport (SN) services.

The services listed above 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 application association 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 of these xDLMS_ASE-s.

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

On the client side, in order to handle the different referencing schemes transparently for the COSEM client application process, 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. Windows98, 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 Logical name 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 application process into/from the service set, used by the server application process. Figure 6 shows the COSEM client application layer when the server is using Short name references. The additional component is called SN_MAPPER_ASE. See also 6.5.5.2.

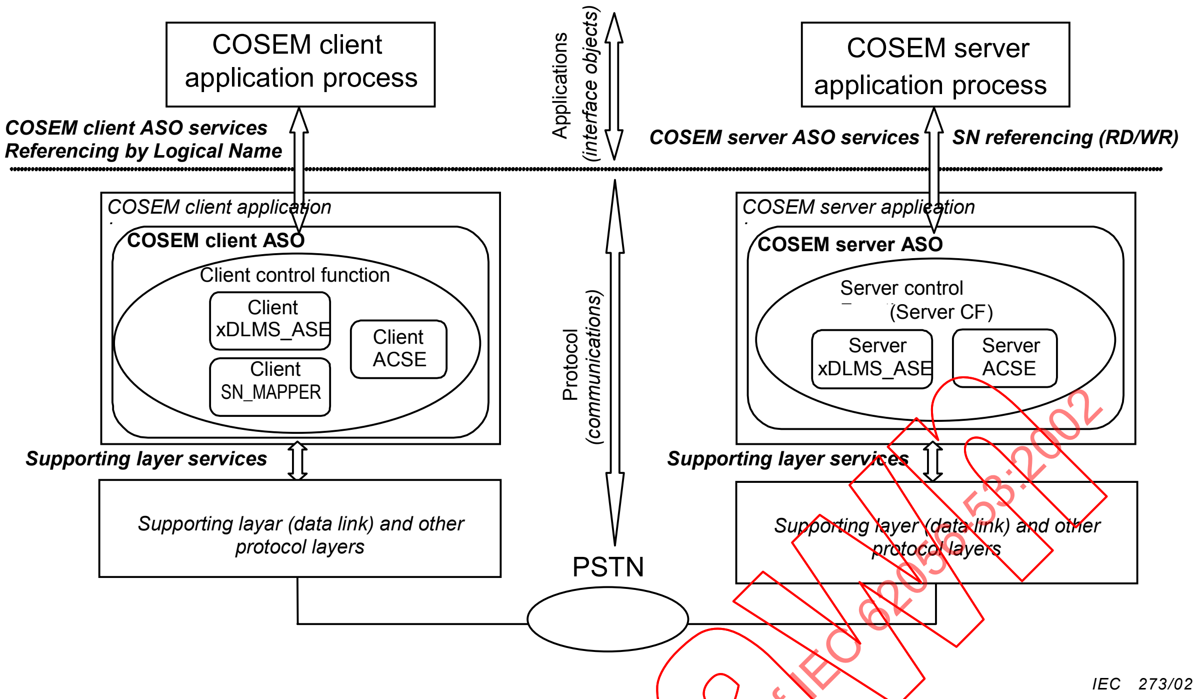

COSEM client application process
COSEM server application process
Applications (interface objects)
COSEM client ASO services
Referencing by Logical Name
COSEM server ASO services
SN referencing (RD/WR)
COSEM client application
COSEM client ASO
Client control function
Client xDLMS_ASE
Client ACSE
Client SN_MAPPER
COSEM server application
COSEM server ASO
Server control (Server CF)
Server xDLMS_ASE
Server ACSE
Protocol (communications)
Supporting layer services
Supporting layer services
Supporting layar (data link) and other protocol layers
Supporting layer (data link) and other protocol layers
PSTN
IEC 273/02

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.

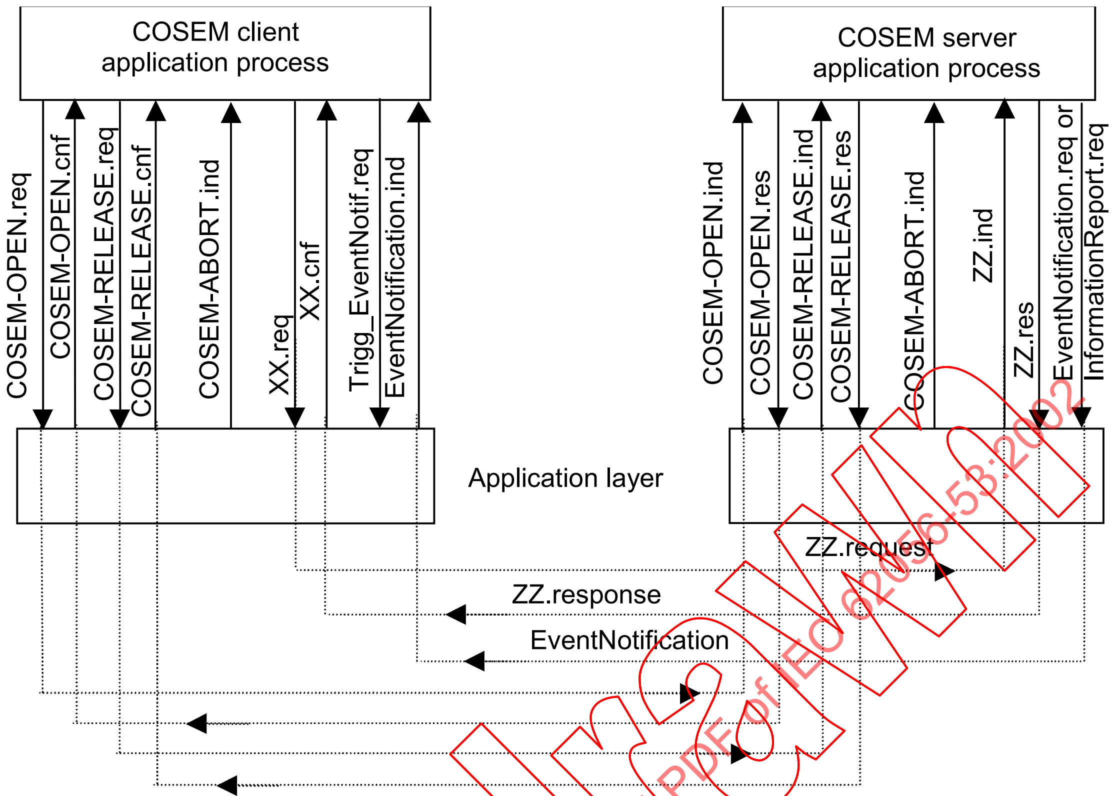


Figure 7 – Summary of COSEM application layer services

NOTE In the figure, above XX and ZZ refers 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.

6.2 Application association establishment and release

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

The COSEM-OPEN.request service is invoked by the COSEM client application process to open an application association to a COSEM server application process. Invoking this service implies – after connecting the lower layers [6] – to generate a COSEM-OPEN.indication service primitive at the server side. The server shall respond to this request by invoking the COSEM-OPEN.response service, which is transferred to the client application process as a remote confirmation (COSEM-OPEN.confirm). This normal opening sequence is shown in Figure 8.

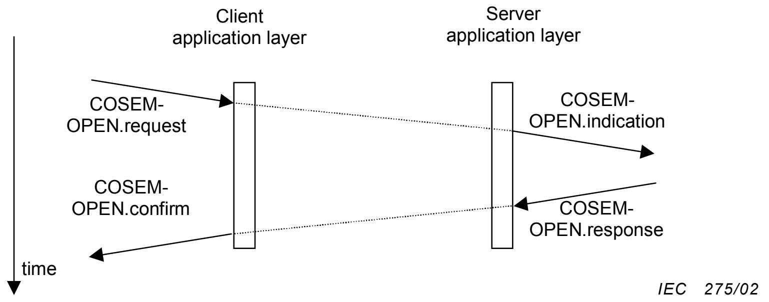


Figure 8 – Normal service sequence for the COSEM-OPEN service

[6] Except for the physical layer, which should be already connected.

NOTE The COSEM-OPEN.request may also be locally 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 is 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 Mandatory application associations

As specified in 4.6 of IEC 62056-62, each physical device shall contain a management logical device. The mandatory contents of the management logical device are defined in 4.6.4 of IEC 62056-62. The management logical device must support an application association to a public client, with the lowest security level. The client address 0x10 is reserved for the public client.

6.3.2 Pre-established application associations

A pre-established application association does not need to be established using the COSEM-OPEN service. It can be considered, that this OPEN has already been done (it is of no importance how). Consequently, pre-established application associations should be considered to be existing from the moment of the establishment of the physical connection between the client and the server devices. A pre-established application association 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 application association eliminates the need of connection establishment and release (phases 1 and 3 on Figure 4) and only data communication services are used. These must use connectionless services of the supporting lower protocol layers.[7]

6.3.3 Non-confirmed application associations

A client application may invoke the COSEM-OPEN.request service in two different ways: confirmed or non-confirmed. A non-confirmed COSEM-OPEN.request invocation shall result in the establishment of a non-confirmed application association. Within this application association the client COSEM application layer shall accept only non-confirmed xDLMS service requests (GET, SET, ACTION). The purpose of having this type of association is to allow multicasting and broadcasting.

6.4 Data communication

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

[7] Pre-established associations cannot be supported by a lower protocol layer not providing non-connected data communication services.

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).

The client application layer obtains knowledge during the application association establishment phase about the referencing method used by the server. When the client application process invokes the data communication services, the COSEM client application layer shall send the APDU corresponding to the appropriate service invocation to the server (referred to as ZZ in Figure 7).

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 application process. In most cases, the server application process 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 application process.

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 application association establishment. These services are provided by the ACSE.

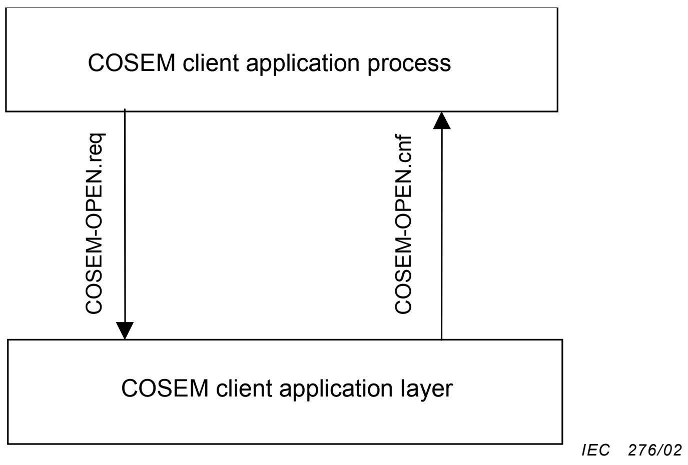

COSEM client application process
COSEM-OPEN.req
COSEM-OPEN.cnf
COSEM client application layer
IEC 276/02

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 application process to request the establishment of an application association to a remote COSEM server application process.

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 service parameter contains all the information necessary to establish lower layer protocol connections. See Annex A.

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 and 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 application process. Default and allowed values for these fields are defined in 7.3.7.

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 confirmed or unconfirmed manner.

Use

The client application process uses this service to initiate the establishment of an application association to a remote server application process.

When the service is invoked with Service_class == Confirmed, the COSEM client shall first establish all required lower layer connections using the service parameters received (except for the physical layer connection, which must be already established prior to this service invocation). Then the COSEM application layer shall send an AARQ APDU to its peer application layer, including the service parameters received.

If the client application process invokes a COSEM-OPEN.request with Service_Class == Confirmed with the same parameters as an already established application association, then the application layer shall locally and negatively confirm this second COSEM-OPEN.request for the reason that the requested application association is already existing.

The COSEM-OPEN.request service with Service_class == Unconfirmed can be used to establish non-confirmed application associations (see 6.3.3). Unconfirmed association is normally used with connectionless profiles – but for special purposes (e.g. multicasting and broadcasting), COSEM allows to establish non-confirmed application associations within connection oriented profiles, too.

The main purpose of invoking the COSEM-OPEN.request service with Service_class == Unconfirmed is to communicate to the client side application layer the necessary parameters – lower layer addresses, application and xDLMS contexts, etc. – for the data communications phase. The service invocation, depending on the implementation, shall or shall not cause the client application layer to send a corresponding AARQ frame[8]. If transmitted, this AARQ is sent using connectionless data services of the supporting lower layer protocol stack. In both cases, the service shall be locally confirmed by the client application layer.

The protocol for application association 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 application association 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,

8 Both behaviours are allowed and are conform to this standard.

ACSE_Protocol_Version,

Application_Context_Name,

Application_Ids_and_Titles,

Security_Mechanism_Name,

Responding_Authentication_Value,

Implementation_Information

)

The Protocol_Connection_Parameters service parameter contains all the information required to identify the protocol connection having been established. These parameters identify the participants of the application association 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 application process 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 application process whether the previously proposed application association 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:

- in case of a pre-established and/or non-confirmed application association;
- if the requested application association already exists;
- due to a locally detected error (missing or not correct parameters, failure during the establishment of lower layer connections or missing physical connection).

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 application association.

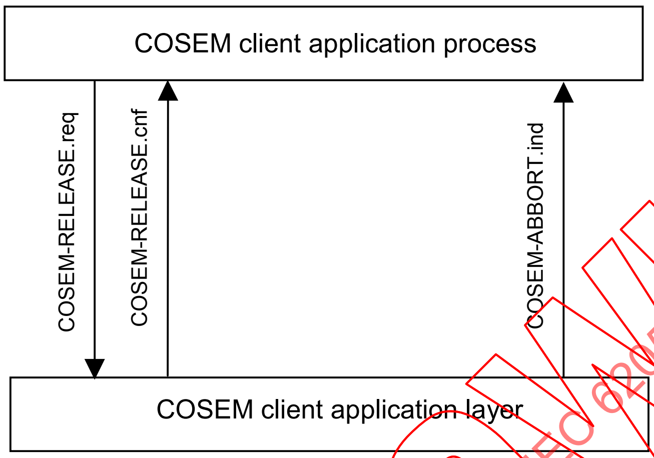

COSEM client application process
COSEM-RELEASE.req
COSEM-RELEASE.cnf
COSEM-ABBORT.ind
COSEM client application layer
IEC 277/02

Figure 10 – Client services for releasing an application association

In the COSEM environment, an application association is unambiguously identified by the corresponding lower protocol layer addresses (SAPs). Therefore, the application association can be non-ambiguously released by disconnecting the appropriate lower connection. Consequently, there is no APDU associated to the RELEASE service: the service is simply accomplished at the lower protocol layer level.

Graceful disconnection – an application/data link level disconnection, which does not imply physical disconnection – can be requested only by the COSEM client application process by invoking the COSEM-RELEASE.request service. This is a remotely confirmed[9] service, and implies a message exchange between the client and server supporting layers.

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

Any existing application association shall be aborted when the physical connection is disconnected. Requesting a physical channel disconnection is done outside of the protocol, therefore it is not within the scope of this standard. A COSEM-ABORT.indication primitive is provided to indicate a non-solicited physical link disconnection to the application process.

6.5.2.2 COSEM-RELEASE.request

Function

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

Service parameters

The semantics of this service primitive is as follows:

COSEM-RELEASE.request

[9] Locally confirmed only when an error condition occurs, for example there is no response from the server.

(

User_Information

)

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 application process uses this service primitive to gracefully release an existing application association with a remote server application process. As an application association is bound to a supporting layer connection on a one-to-one basis, the invocation of this service shall not imply sending an APDU. Instead, the COSEM client application layer shall initiate the disconnection of the corresponding lower layer connection by invoking the corresponding XX-DISCONNECT.request service of the supporting lower protocol layer.

The protocol for releasing an application association 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 application process whether the previously received request for releasing the application association is accepted.

NOTE The server cannot refuse a release request.

Service parameters

The semantics of the primitive are as follows:

COSEM-RELEASE.confirm

(

Result,

Failure_type,

User_Information

)

In case of a remote confirmation, the Result parameter indicates the success or the failure of the corresponding association release request. In case of a local confirmation, the Result parameter is always set to ERROR.

In the case of non-acceptance, the Failure_type indicates the reason for the failure.

NOTE This parameter is generated locally.

The User_Information field may be present only when the association 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 application process the result of the previously requested release of an application association. This service primitive shall be originated either 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.

6.5.2.4 COSEM-ABORT.indication

Function

This service is invoked by the client application layer to indicate to the client application process an unsolicited 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 physical 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 application process that a physical connection abort occurred in a non-solicited manner (e.g. the physical line is cut).

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.

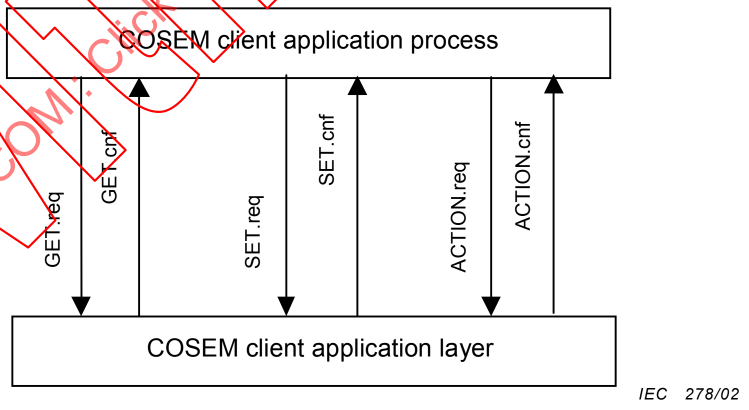

COSEM client application process
GET.req
GET.cnf
SET.req
SET.cnf
ACTION.req
ACTION.cnf
COSEM client application layer
IEC 278/02

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 application process.

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 application process. The role of the protocol with regard to these services is to transport them as .indication to the COSEM server application process.

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 application association can be invoked in confirmed or non-confirmed manner. In case of non-confirmed application associations, data communication services may only be invoked in a non-confirmed manner.

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

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 application process to request the value(s) of one or all attributes of one or more COSEM interface object(s) from the remote server application process.

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 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 service parameters, each of them referencing one or all attributes of a COSEM interface object. The COSEM_Attribute_Descriptor service 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 application process 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 application process.

6.5.3.3 GET.confirm

Function

This service is invoked by the client application layer to indicate the reception of a GET.response APDU from the COSEM server application process.

Service parameters

The semantics of the primitive is as follows:

GET.confirm

(

Invoke_Id,

Priority,

Response_type,

Result, { Result , }

Block_Number

)

The Invoke_Id 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 APDU.

6.5.3.4 SET.request

Function

This service primitive is invoked by the client application process to request the remote server application process 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 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 and high.

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. This parameter 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 shall be generated.

A SET.request service shall contain one or more COSEM_Attribute_Descriptor service parameters, each of them referencing one or more attributes of a COSEM interface object. The COSEM_Attribute_Descriptor service 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 application process uses this service primitive in order to request the remote server application process 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 application process.

Service parameters

The semantics of the primitive is as follows:

SET.confirm

(

Invoke_Id,

Priority,

Response_type,

Result { Result, },

Block_Number

)

The Invoke_Id 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 APDU.

6.5.3.6 ACTION.request

Function

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

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 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 and high.

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

The Request_type parameter indicates whether the given invocation contains a complete request or only a part of it. This parameter 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 service parameters, each of them referencing one method of a COSEM interface object. The

COSEM_Method_Descriptor service 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 application process to remotely invoke one or more method(s) of one or more COSEM interface object(s) in the remote server application process.

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 application process.

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 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 then 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 do not fit in 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

The client application layer uses this service primitive to indicate the reception of a ACTION.response APDU.

6.5.4 Client side services for event notification

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

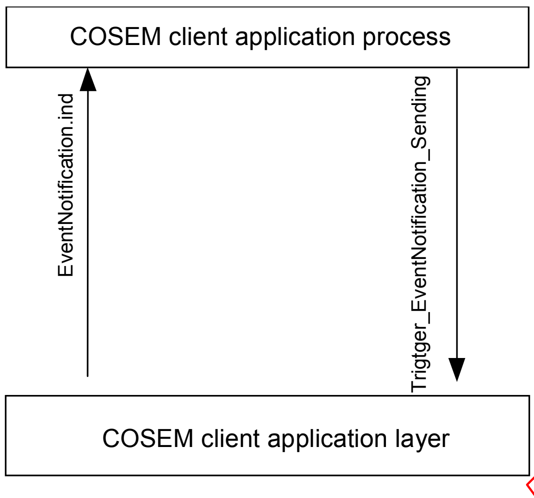

COSEM client application process
EventNotification.ind
Trigtger_EventNotification_Sending
COSEM client application layer
IEC 279/02

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 application process is able to send an unsolicited notification of the occurrence of an event to the remote client application. Reception of the EventNotification message is indicated to the client application process 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 EventNotification.indication from the COSEM server application process.

Service parameters

The semantics of the primitive is as follows:

EventNotification.indication

(

Time,

Protocol_Parameters,

COSEM_Attribute_Descriptor,

Attribute_Value

)

COSEM_Attribute_Descriptor

(

COSEM_Class_Id,

COSEM_Object_Instance_Id,

COSEM_Object_Attribute_Id

)

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

The Protocol_Parameters service parameter contains all protocol parameters, which are required to identify the source and the destination application processes of the EventNotification.indication message.

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 service 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 EventNotification.indication to the client application process.

6.5.4.2 Trigger_EventNotification_Sending.request

Function

This service is invoked by the client application process in order to trigger the server to send the frame carrying the Event-Notification-Request APDU.

NOTE This service is necessary in case of lower layer protocols 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 service 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 application process, 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 application process 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 application association. The structure of the content is defined in IEC 62056-62.

Use

The client application process 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 (comp. IEC 61334-4-41) by the client Control function (see Figure 6) in the following manner:

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 application association establishment. These services are provided by the ACSE.

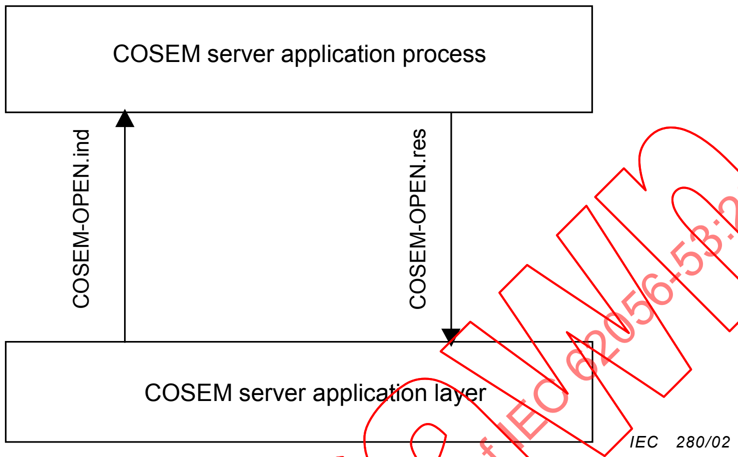

COSEM server application process
COSEM-OPEN.ind
COSEM-OPEN.res
COSEM server application layer
IEC 280/02

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 application process that the peer (client) application process requested the establishment of an application association.

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 service parameter contains all the information necessary to establish lower layer protocol connections. See Annex A.

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 Application Process. 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 Application Process 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 application process. All lower layer connections have to be already established.

The protocol for application association establishment is described in 7.3.1.

6.6.1.3 COSEM-OPEN.response

Function

This service is invoked by the server application process to indicate whether the previously proposed application association 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 service parameter contains all the information required to identify the protocol connections having been established.

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

In the case of non-acceptance, the Failure_type parameter indicates the reason for not accepting the proposed application 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.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 application process to indicate to the application layer whether the previously proposed application association is accepted or not. This primitive is invoked only, if the COSEM-OPEN.indication has been invoked in a confirmed manner.

6.6.2 Application association release

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

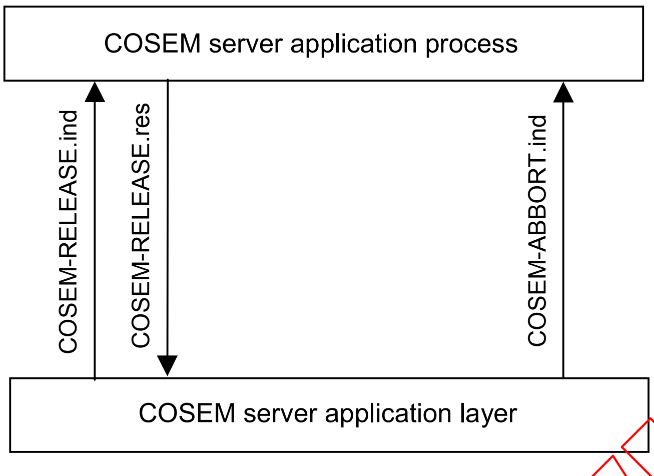

COSEM server application process
COSEM-RELEASE.ind
COSEM-RELEASE.res
COSEM-ABBORT.ind
COSEM server application layer
IEC 281/02

Figure 14 – Server side services for releasing an application association

In the COSEM environment, an application association is unambiguously identified by the corresponding lower layer addresses (SAPs). Therefore, it can be non-ambiguously released by disconnecting the appropriate lower layer connections. Consequently, there is no APDU associated to the RELEASE service: the service is simply accomplished at the lower protocol layer level.

Graceful disconnection – an application/data link level disconnection, which does not imply physical disconnection – can be requested only by the COSEM client application process by invoking the COSEM-RELEASE.request service. This is a remotely confirmed[10] service and implies a message exchange between the client and the server supporting layers. The server application responds to a COSEM-RELEASE.indication service with the invocation of the COSEM-RELEASE.response service primitive.

Upon the reception of this response service primitive, the client application layer shall confirm the preceding .request service with the COSEM-RELEASE.confirm service primitive to the client application process.

Any existing application association shall be aborted when the physical connection is disconnected. Requesting a physical channel disconnection is done outside of the protocol, therefore it is not within the scope of this standard. A COSEM-ABORT.indication primitive is provided to indicate a non-solicited physical link disconnection to the application process.

6.6.2.1 COSEM-RELEASE.indication

Function

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

Service parameters

The semantics of the primitive is as follows:

COSEM-RELEASE.indication

(

User_Information

)

[10] Locally confirmed only if an error condition occurs, for example there is nothing received as the response for the DISC frame.

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 upon the reception of a supporting layer disconnect indication with REASON == REMOTE, to indicate to the server application process that a graceful release of the application association has been requested. The server must accept this request.

6.6.2.2 COSEM-RELEASE.response

Function

This service primitive is invoked by the COSEM server application process to indicate to the application layer whether the previously received request for releasing the application association 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

)

The Result parameter indicates whether the server application process can accept the previous COSEM-RELEASE.request or not. Its value depends on whether the application association, 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.

Use

This service primitive is used by the server application process. Upon the invocation of this service primitive, the server application layer shall invoke the .response service primitive of the supporting layer, with the appropriate OK, NOK or NO_RESPONSE Result parameter.

6.6.2.3 COSEM-ABORT.indication

Function

This service is invoked by the server application layer to indicate to the server application process an unsolicited 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 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 upon the reception of a supporting layer disconnect indication service primitive with REASON == LOCAL, indicating that a physical connection abort occurred in a non-solicited manner (e.g. the physical line is cut).

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 application association, 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 association 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:

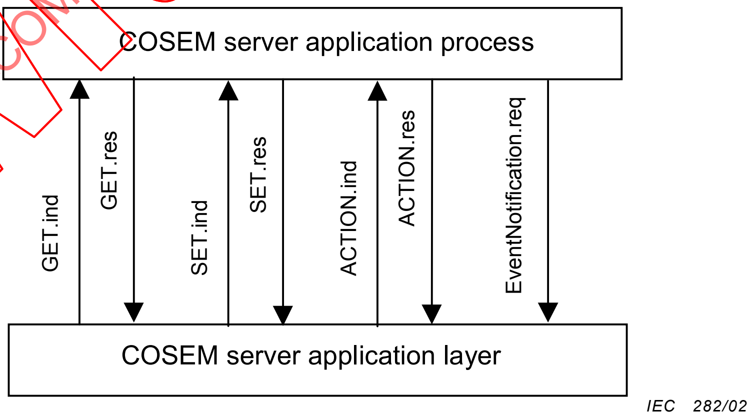


Figure 15 – Server side data communications services using LN referencing

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 application process. The role of the protocol with regard to these services is to transport them to the COSEM server application process. The server application layer shall indicate the reception of a request via the .indication service primitive to the server application process.

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

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

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 application process 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 application process 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 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 and high.

The Service_Class parameter indicates whether the service is invoked in confirmed or 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 service parameters, each of them referencing a COSEM interface object attribute. The COSEM_Attribute_Descriptor service 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 application process 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 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 application process. 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 application process 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 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 and high.

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 a complete attribute or only a block of it. This parameter 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 service parameters, each of them referencing one or all attributes of a COSEM interface object. The COSEM_Attribute_Descriptor service 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 application process 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 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 application process. Upon the reception of the SET.response service invocation, the COSEM server application layer shall build a SETresponse 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 application process 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 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 and high.

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

The Request_type parameter indicates whether the given invocation contains a complete request or only a part of it. This parameter 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 service parameters, each of them referencing one COSEM interface object method. The COSEM_Method_Descriptor service 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 application process 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 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 then 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 application process. Upon the reception of the ACTION.response service invocation, the COSEM server application layer shall build an ACTION.response APDU. In case of success – when the corresponding ACTION.indication has been accepted – this APDU shall contain only a positive acknowledgement, and – if the requested ACTION has to return data, fitting in one APDU – the data to be returned.

When the required data does not fit into one APDU, – similarly to the GET.response service – it is sent back to the client in data blocks, with the help of the transparent or non-transparent long data transfer mechanism. These mechanisms are defined in 7.4.1.8.

In case of failure a negative acknowledgement is sent, indicating whether the required action could not be accepted (action-error), or the required data cannot be accessed (data-access-error).

In both 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 application process to send an EventNotification message to the remote client application process.

Service parameters

The semantics of the primitive is as follows:

EventNotification.request

(

Time,

COSEM_Attribute_Descriptor,

Attribute_Value

)

COSEM_Attribute_Descriptor

(

COSEM_Class_Id,

COSEM_Object_Instance_Id,

COSEM_Object_Attribute_Id

)

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

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 service parameter carries the value of this attribute. More information about the notified event may be obtained by interrogating this COSEM interface object.

The EventNotification.request service invocation shall not contain protocol information: it shall always be sent from the address of the server management logical device to the client management application process. Both application processes are always present and in any protocol profile they are bound to a fix address.

Use

This service is used by the server application process. Upon the reception of the EventNotification.request service invocation, the COSEM server application layer shall build the Even-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 IEC 61334-4-41.

6.6.3.3.3 WriteRequest

The service is described in 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.

7 COSEM application layer protocol specification

The COSEM application layer is based on the extended DLMS – xDLMS, see Annex B – and on the standard 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/TR2 8650-1 respectively.

Both the xDLMS and the application contexts can be negotiated during the application association 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).

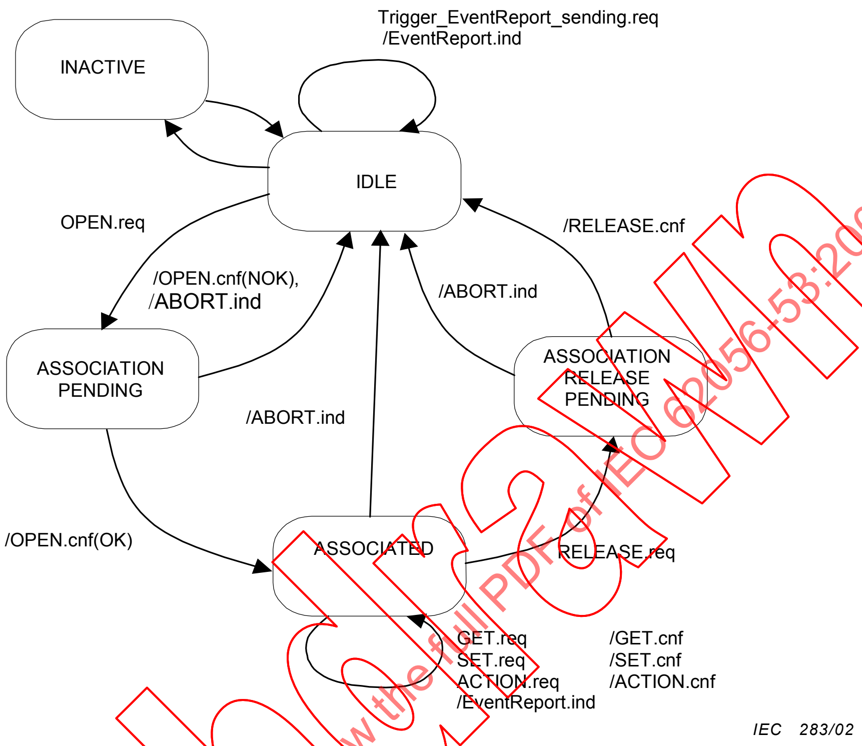


Figure 16 – Partial state machine for the client side control function

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.

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 application process 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 application association created, being released or currently established[11]. 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 application process invokes the COSEM-OPEN.request (OPEN.req)

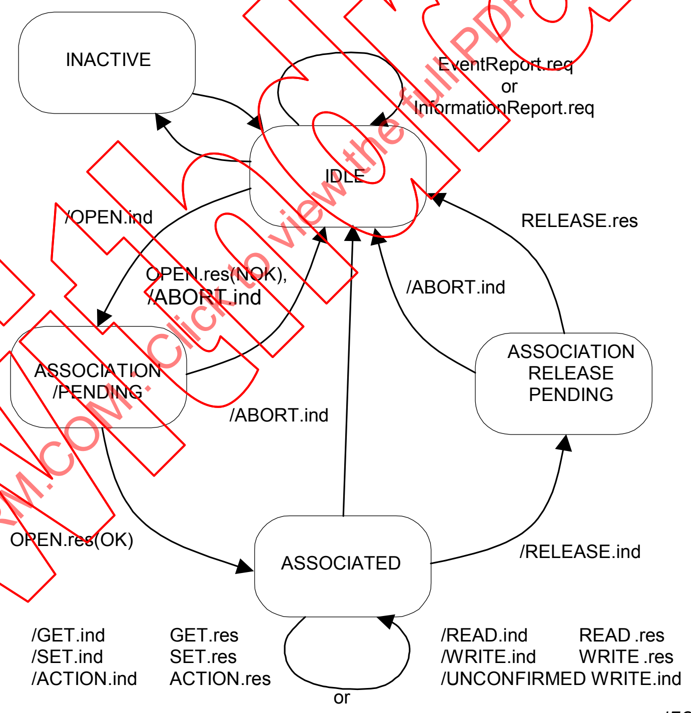
[11] Note, that it is the state machine for the application layer: lower layer connections, including the physical connection, are not taken into account. On the other hand, physical connection establishment is done outside of the protocol.

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 application process. Depending on the result of the association request, the client CF shall return to IDLE state (NOK), or shall enter the ASSOCIATED state;

- ASSOCIATED – the CF shall enter this state when the application association 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 the association is aborted due to a non-solicited physical disconnection;
- 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 disconnection of the established application association. 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 the release response from the remote server or by the reception of a DL-DIS-CONNECT.indication service primitive meaning that the physical layer has been aborted.

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.



IEC 284/02

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 application process 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 application association 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 application process 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 application process. 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 application association 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 association by invoking the COSEM-RELEASE.request service (/RELEASE.ind), or the association is aborted due to a non-solicited physical disconnection;
- ASSOCIATION RELEASE PENDING – upon the reception of a COSEM-RELEASE.request service primitive from the remote client application process, the CF of the application layer protocol entity shall indicate it to the application process (/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 also originated by the reception of a DL-DISCONNECT.indication service primitive, meaning that the physical layer has been aborted.

7.3 Protocol for application association establishment/release

7.3.1 Establishment of an application association

Application association establishment with the help of the Association.request/ .indication./ .response/ .confirmation services of the standard ACSE, ISO/IEC/TR2 8650-1 is the key element of COSEM interoperability. The participants of an application association are the interoperable communications partners:

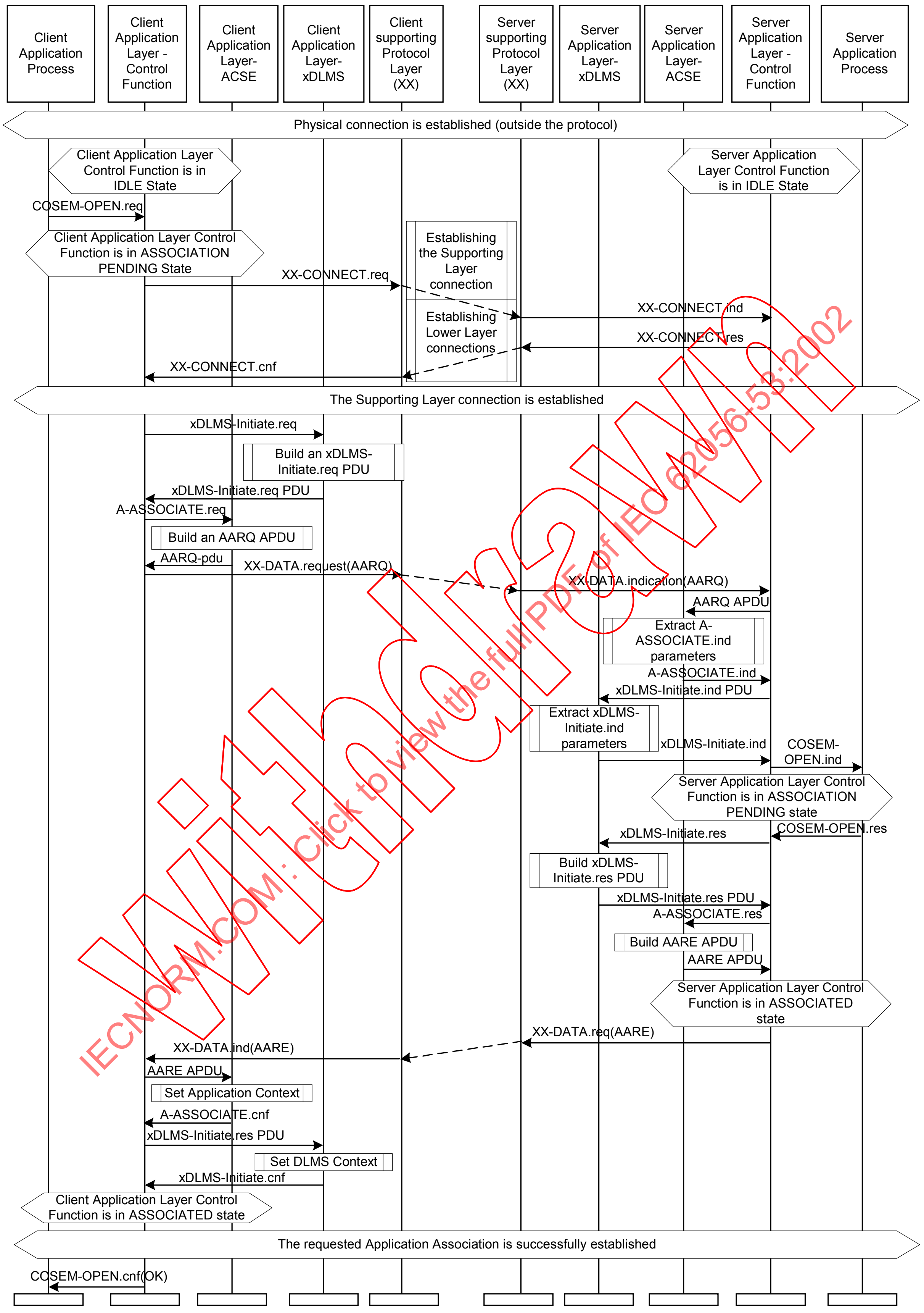
- a client application process, which is always the originator of an application association request, and;
- a server application process[12].

The client application process shall first[13] 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 establish the required lower layer connections.

Supposing that there is no problem, the two supporting layers are connected, as the MSC shows in Figure 18.

[12] 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.

[13] Invoking the COSEM-OPEN.request service requires an already established physical connection, but the establishment of this physical connection takes place outside the protocol.



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.req(AARE)
XX-DATA.ind(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 285/02

Figure 18 – MSC for successful application association establishment

Once the 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[14], to the COSEM server application process. At the same time, the server Control function shall enter the 'ASSOCIATION PENDING' state.

The server application process shall analyze the received COSEM-OPEN.indication primitive, and decide whether it accepts the proposed application associations or not[15]. Following this verification, the COSEM server application process shall invoke the COSEM-OPEN.response service to indicate the acceptance or non-acceptance of the proposed association. In case of success, the CF shall assemble the appropriate AARE APDU, then shall send it to the remote client application layer via the existing supporting layer connection, and the server application layer shall enter the 'ASSOCIATED' state. From this moment, the server is able to receive data communication service .request(s) and send .responses within this association. 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 application process via the COSEM-OPEN.confirm service primitive. At the same time, the client application layer shall enter the 'ASSOCIATED' state. From this moment, the application association 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 application associations need not to be established using the COSEM-OPEN service. This standard does not specify the way of establishing these associations. Pre-established associations should be considered to exist when the physical connection is established between the client and the server devices.

A pre-established association can be either confirmed or non-confirmed, depending on the way it is pre-established.

7.3.2.2 Establishment of non-confirmed application associations

A non-confirmed COSEM-OPEN.request invocation shall result in the establishment of a non-confirmed application association. Within this application association, the client COSEM application layer shall accept only non-confirmed xDLMS service requests (GET, SET, ACTION). The main purpose of having this type of association is to allow multicasting and broadcasting.

[14] Some service parameters of this COSEM-OPEN.indication primitive (address information, User_Information, Service_Class) do not come from the AARQ APDU, but from the supporting layer frame carrying the AARQ APDU.

[15] 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.

A non-confirmed COSEM-OPEN.request invocation may be locally confirmed in the case of a locally detected error.

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[16]. 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
  }
```

[16] 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;

NOTE 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 application association, and these fields shall also 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 the server is expected to use. In the AARE, the mechanism name defines the authentication mechanism required by the server, i.e. the mechanism which the client is expected to use.

- **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;

NOTE In the COSEM environment, the response-allowed parameter of the xDLMS-Initiate.request PDU shall always be set to TRUE.

- **result**: this parameter carries the result of the proposed application association 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/TR2 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 application association 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 B.

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 application association 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, service class;

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.

If the server refuses the proposed application association with the reason of non-fit at the COSEM application context level, it shall send back an AARE APDU, containing diagnostics information about the failure. In this case, the user-information field of the response AARE APDU shall contain the server's own xDLMS context: the supported DLMS version number, the supported conformance block and the server-max-receive-pdu-size.

If the proposed COSEM application context is acceptable, the server shall check the proposed xDLMS context. The parsing order shall be the following:

a) proposed-dlms-version-number

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

If this context is also acceptable, the server shall accept the proposed association and 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.

In this case, when the AARE contains an xDLMS-Initate.response PDU, 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, with "no-reason-given" as diagnostics information. The user-information field 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 handling of a COSEM-OPEN.request invocation with the parameters of an already established AA depends on the type of the newly requested AA and the type of this existing AA.

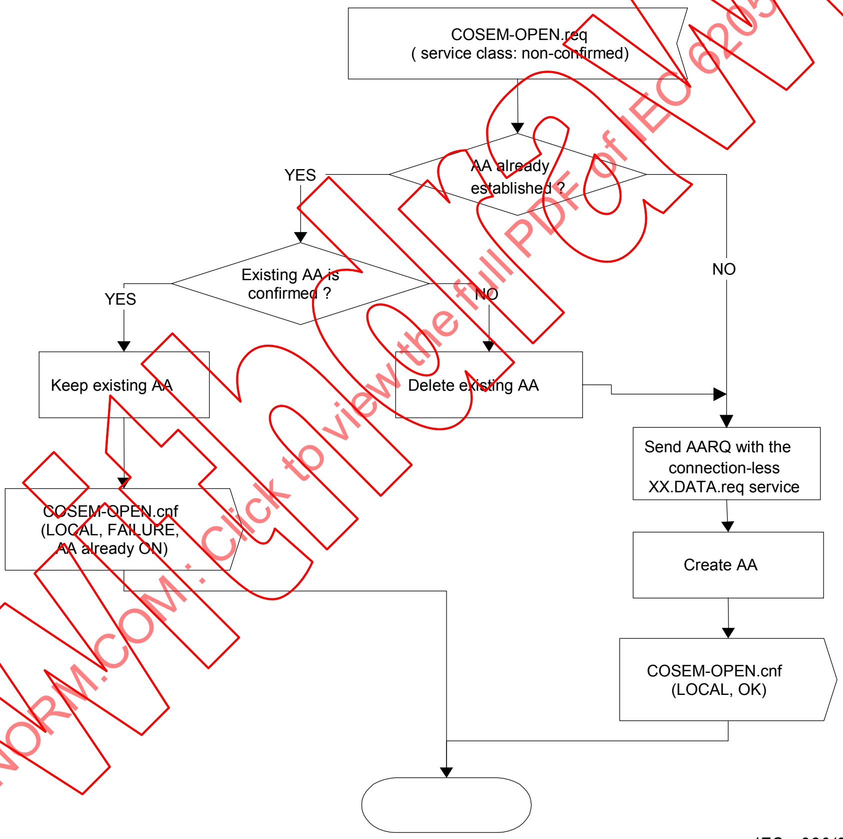
7.3.5.1 Handling repeated non-confirmed COSEM-OPEN.requests

7.3.5.1.1 Client side

If the existing AA has been established with the help of a confirmed COSEM-OPEN.request, the newly invoked non-confirmed OPEN.request shall not imply any action towards the server. The previously established association shall be kept as it is, and the application layer shall locally (and negatively) confirm the second COSEM-OPEN.request with a COSEM-OPEN.confirm, indicating that the COSEM-OPEN.request has been locally failed because an AA is already existing.

If the existing AA has been established with the help of a non-confirmed COSEM-OPEN.request, the client application layer shall issue an AARQ with the parameters of the new COSEM-OPEN.request, and shall replace the previous AA with the new one.

The flow chart on Figure 19 summarizes non-confirmed COSEM-OPEN.request handling at the client side.


COSEM-OPEN.req
(service class: non-confirmed)
AA already established ?
YES
NO
Existing AA is confirmed ?
YES
NO
Keep existing AA
Delete existing AA
Send AARQ with the connection-less XX.DATA.req service
Create AA
COSEM-OPEN.cnf (LOCAL, FAILURE, AA already ON)
COSEM-OPEN.cnf (LOCAL, OK)

IEC 286/02

Figure 19 – Handling non-confirmed COSEM-OPEN.request at the client side

If the existing AA is a pre-established AA, the second COSEM-OPEN.request shall not imply any action towards the server. The pre-established association shall be kept as it is, and the application layer shall locally (and negatively) confirm the second COSEM-OPEN.request with a COSEM-OPEN.confirm, indicating that the COSEM-OPEN.request has been locally failed because of an already existing pre-established AA.

7.3.5.1.2 Server side

If the server receives an AARQ containing the parameters of an already established AA, transported within a connectionless XX-DATA.indication service of the supporting lower protocol layer (meaning that it is a non-confirmed COSEM-OPEN.request), the server behaviour shall depend on the type of the already existing AA.

If this, existing AA is a non-confirmed AA, the existing AA shall be deleted (released), and the server shall try to establish the AA corresponding to the newly received request. Even if it is not ensured that this new AA can be created, (e.g. the received AARQ does not contain the right password) the previous AA shall be released.

On the other hand, if the received AARQ indication contains the parameters of an already established confirmed AA, the received AARQ shall be simply discarded, and the existing AA shall be kept as it is.

The flow chart of Figure 20 summarizes non-confirmed COSEM-OPEN.indication handling at the server side.

Finally, if the existing AA is a pre-established AA, the second COSEM-OPEN.request shall not imply any action: the pre-established association shall be kept as it is, and the received AARQ shall be discarded (no COSEM-OPEN.indication shall be generated).

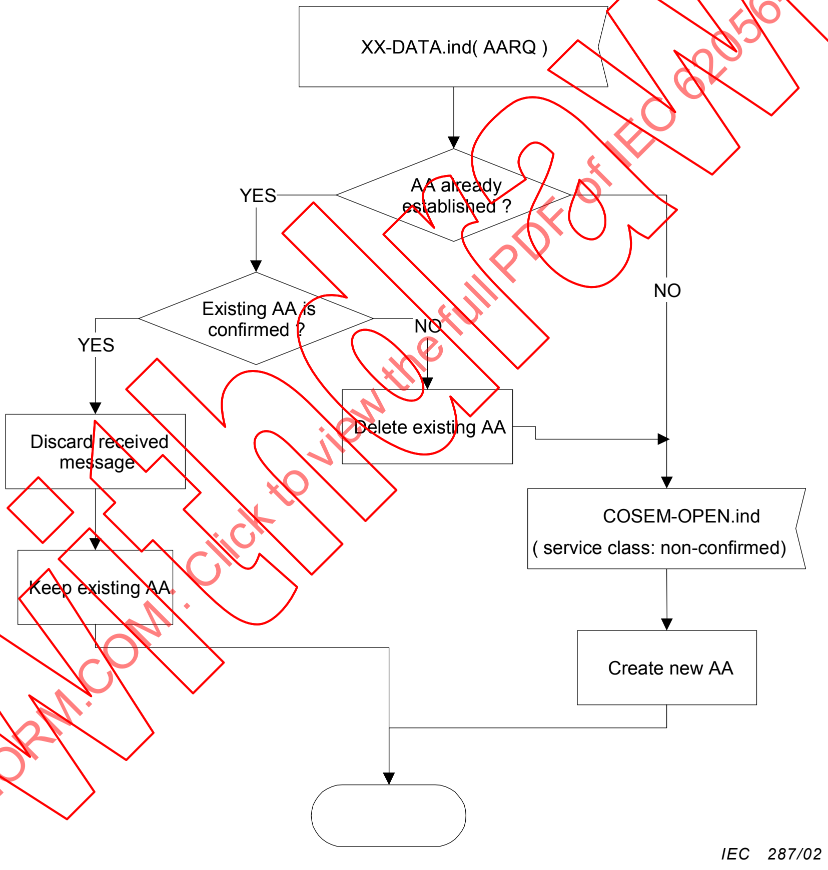


Figure 20 – Handling the reception of a non-confirmed AARQ at the server side

7.3.5.2 Handling repeated confirmed COSEM-OPEN.requests

7.3.5.2.1 Client side

If the existing AA has been established with the help of a confirmed COSEM-OPEN.request, the newly invoked confirmed COSEM-OPEN.request shall not imply any action towards the server: the previously established association shall be kept as it is, and the application layer shall locally (and negatively) confirm the second COSEM-OPEN.request with a COSEM-OPEN.confirm, indicating that the COSEM-OPEN.request has been locally failed because of an already existing AA.

If the existing AA has been established with the help of a non-confirmed COSEM-OPEN.request, the client application layer shall try to establish the newly requested AA, by establishing the required lower layer connection(s) and then sending the AARQ APDU to the server with the help of the connection-oriented XX-DATA.request service. In this case, if the establishment of the newly requested confirmed AA fails (for any reason) the previously existing non-confirmed AA shall be kept. Otherwise – if the establishment of the newly requested AA is successful – the previously existing non-confirmed AA shall be replaced with the new, confirmed AA.

Finally, if the newly requested AA corresponds to a pre-established AA, the second COSEM-OPEN.request shall not imply any action towards the server. The pre-established association shall be kept as it is, and the application layer shall locally (and negatively) confirm the second COSEM-OPEN.request with a COSEM-OPEN.confirm, indicating that the COSEM-OPEN.request has been locally failed because of an already existing pre-established AA.

7.3.5.2.2 Server side

Reception of an AARQ in a confirmed way is possible only on an already established lower protocol layer connection. If the received AARQ indication contains the parameters of an already established AA, the server behaviour shall depend on the type of this already existing AA. If it has been established in a confirmed manner, the reception of the AARQ shall be considered as an error: the existing AA shall be kept and the received AARQ shall be discarded.

In the case when the already existing AA has been established in a non-confirmed manner, the received AARQ shall be handled normally. The parameters of the AARQ shall be decoded and the server application layer shall generate a COSEM-OPEN.indication service primitive with these parameters. If the COSEM application process decides that the requested AA can be accepted, the server shall replace the previously existing non-confirmed type AA with the new one. Otherwise, if the requested AA cannot be accepted, the previously established non-confirmed AA shall be kept. In both cases, the server shall send back an AARE APDU with the appropriate information to the client.

Finally, if the newly requested AA corresponds to a pre-established AA, the second COSEM-OPEN.request shall not imply any action: the pre-established association shall be kept as it is, and the received AARQ shall be discarded (no COSEM-OPEN.indication shall be generated).

7.3.6 Releasing an application association

7.3.6.1 Overview

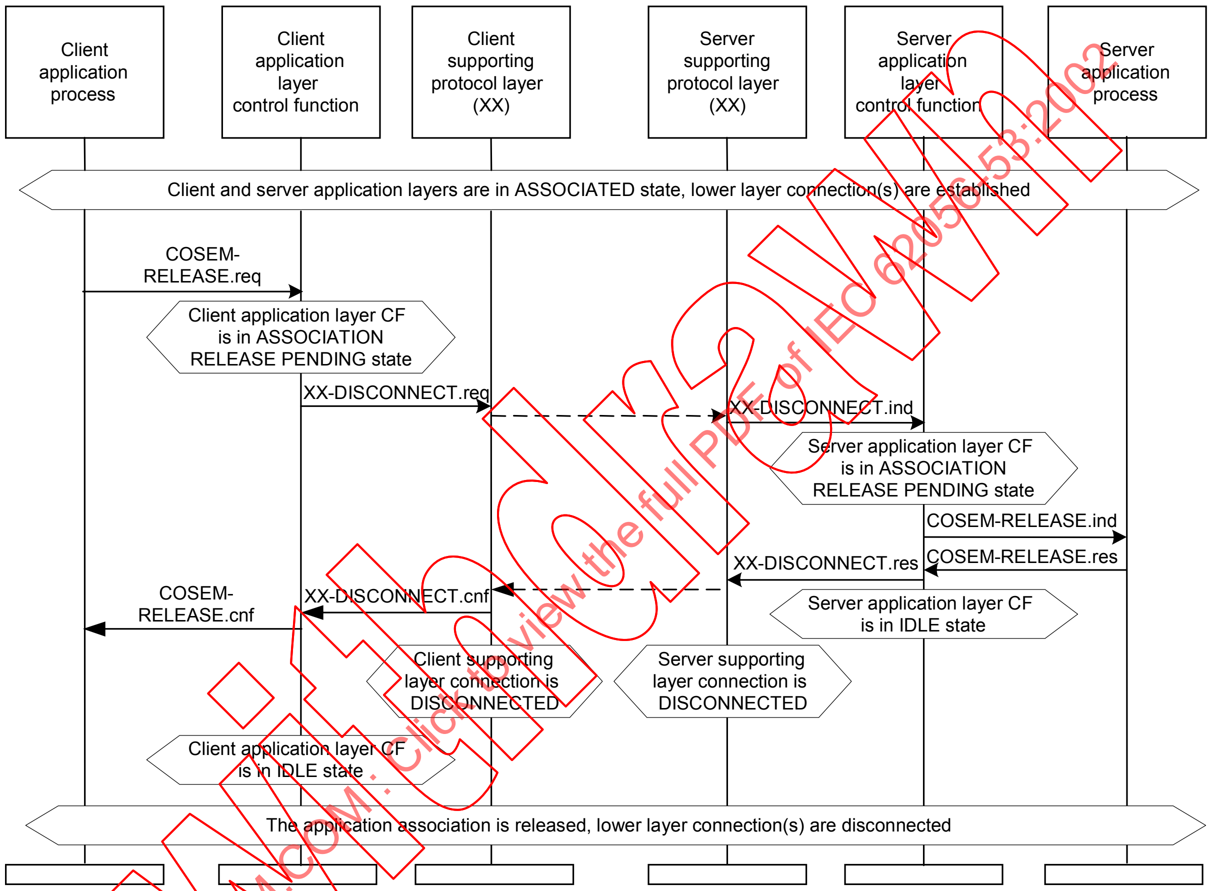
An existing application association can be released gracefully or non-gracefully. Graceful release means that it is the protocol machine, which notifies its peer that it is releasing the association. Graceful release can be initiated only by the client application process.

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 application process.

7.3.6.2 Graceful release of an application association

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

According to the protocol of the ACSE, an A_RELEASE.request APDU should be generated. However, as in the COSEM environment the existence of an application association is bound to the corresponding lower protocol layer connection on a one per one basis, the invocation of the COSEM-RELEASE.request service shall imply directly the invocation of a 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, as it is shown in Figure 21.



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

Figure 21 – Graceful release of an application association

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

The COSEM server application process 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 service parameters. At the same moment, the Control function of the server application layer shall enter the 'IDLE' state[17].

[17] 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.

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 application process 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.

7.3.6.3 Non-graceful release of an application association

Non-graceful release of application associations in COSEM may be the result of a detected physical disconnection. Disconnection of the physical connection can be requested voluntarily by the client or by the server (outside of the protocol), or may be the result of an external event. In both cases, the detection of a physical disconnection is indicated to the Control function of the application layer on both sides with the help of the XX-DISCONNECT.indication service of the supporting protocol layer.

Figure 22 shows the message sequence chart for aborting the application association.

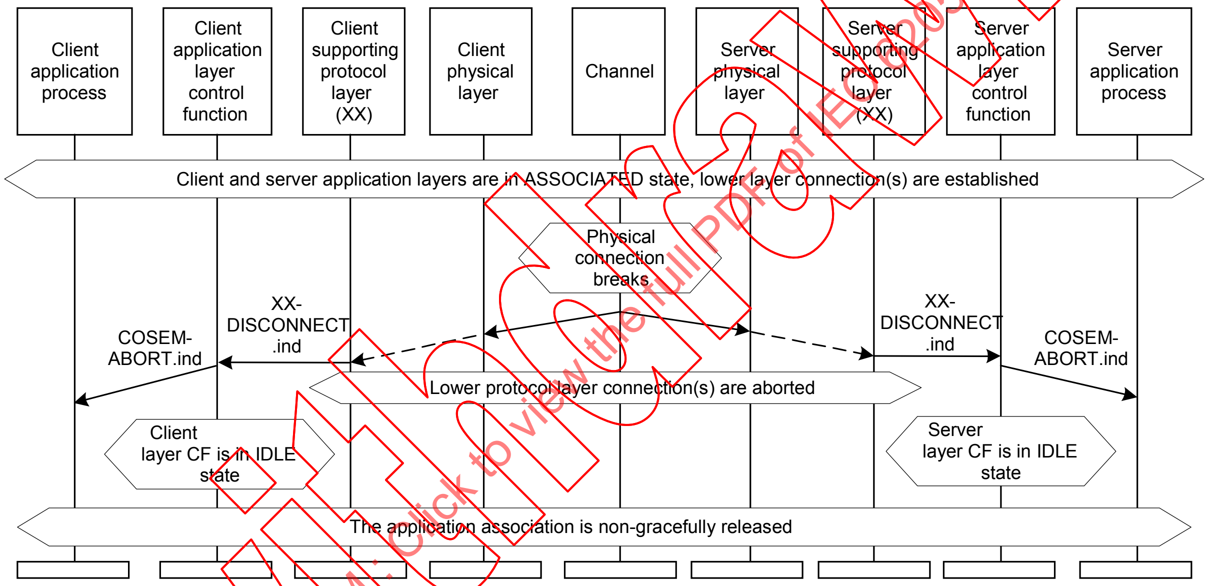

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Client physical layer
Channel
Server physical layer
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s) are established
Physical connection breaks
COSEM-ABORT.ind
XX-DISCONNECT .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 289/02

Figure 22 – Aborting an application association following a PH-ABORT.indication

NOTE The non-graceful release of application association 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 application association, 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 application association.

The application context that applies to an application association is determined during its establishment[18]. 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 application association.

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)}

[18] An application association has only one application context. However, the set of rules that make up the application context of an application association may contain rules for alteration of that set of rules during the lifetime of the application association.

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[19];
- 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.

In order to establish one of these default application contexts, the ACSE AARQ and the AARE APDUs shall carry one of the above values.

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 three authentication mechanism names specified:

default-COSEM-lowest-level-security-mechanism-name[20] ::=
{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)}

[19] With the COSEM extensions to DLMS, see Annex B.

[20] This mechanism is used for client identifier purposes in the case of multicasting and broadcasting.

```
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 is registered by the DLMS UA.

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

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 confirmed or 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 EventNotification service. By requesting this service, the server application process 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 application association only.

The COSEM server application process, 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 application process shall generate the appropriate .response message.

Figure 23 shows a complete message sequence chart for a confirmed GET.request service invocation in case of success.

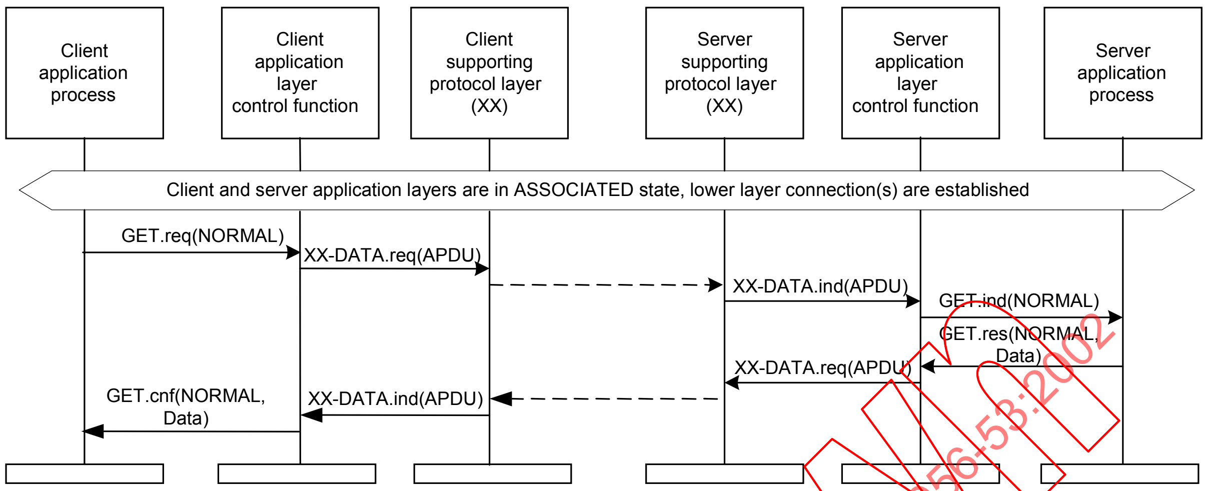

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s) 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)

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

IEC 290/02

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

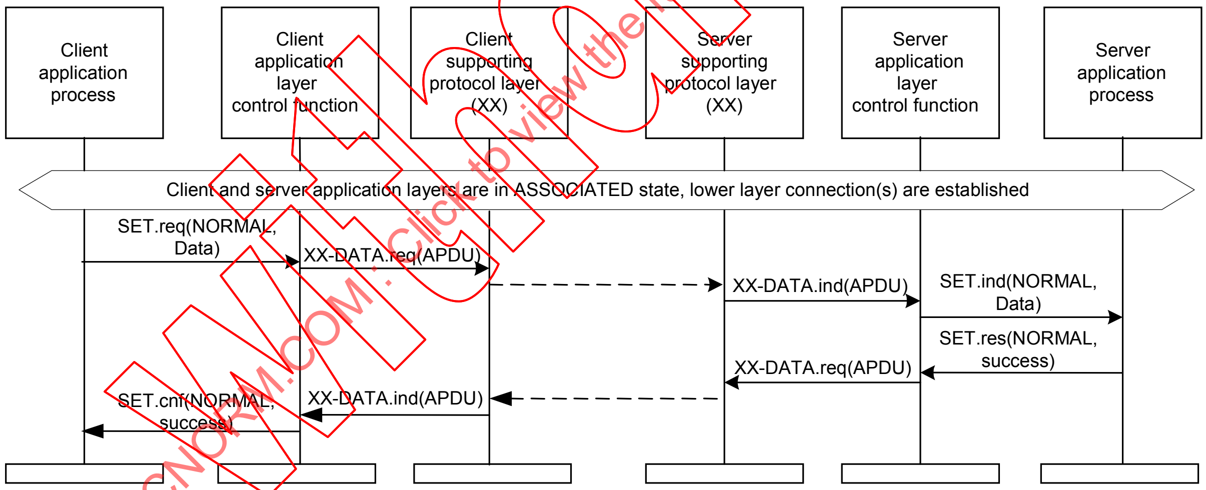

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s) 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)

Figure 24 – MSC for a confirmed SET service in case of success

IEC 291/02

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 25.

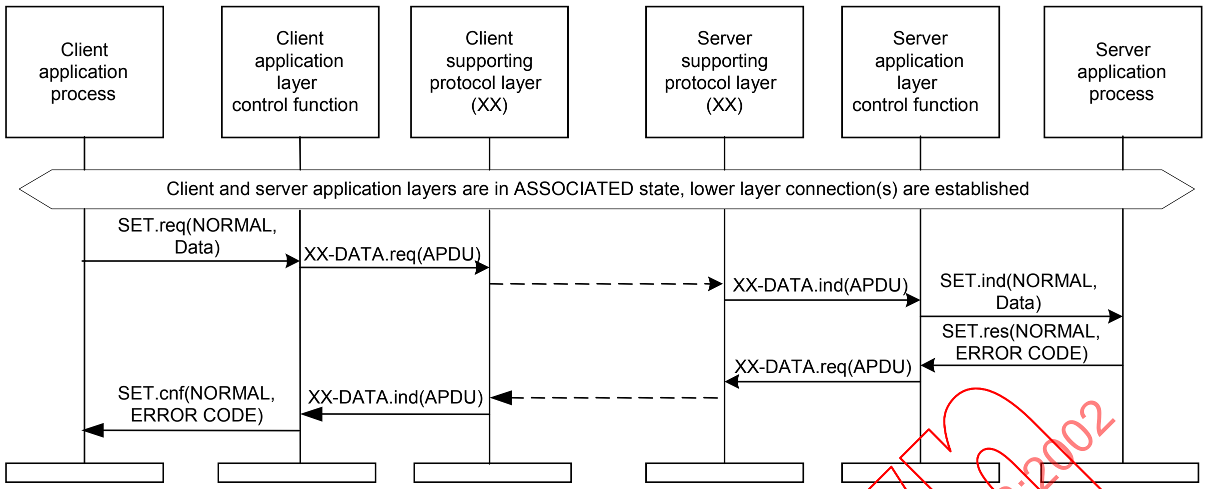

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s)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

Figure 25 – 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 at 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 this data may be too long to fit into one APDU.

Figure 26, 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.

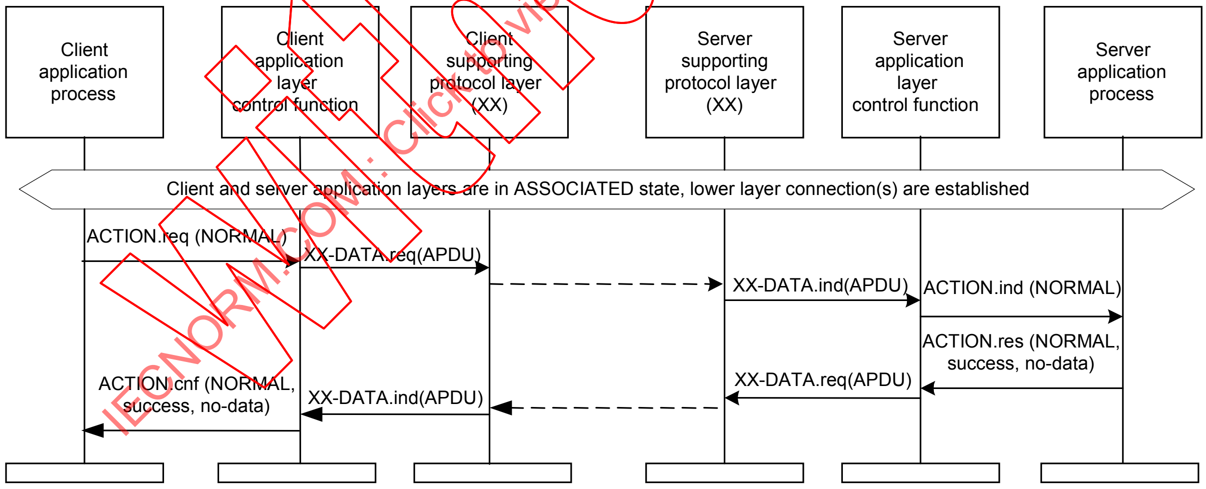

Client application process
Client application layer control function
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer control function
Server application process
Client and server application layers are in ASSOCIATED state, lower layer connection(s)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

Figure 26 – 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 unconfirmed manner within an established confirmed or unconfirmed application association. The following service primitives are supported:

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

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, like alarms, fraud detection, or simply a counter overflow generally occur asynchronously to any operation. As the server can send information only upon a request from the client, the client may or may not gain knowledge about these events using COSEM client/server services.

In order to ensure that the client is informed about such events, a special, non-client/server type service, the EventNotification[21] service is available. Although any logical device in a server device may have events to be reported, event notification in COSEM is the responsibility of the management logical device.

A server device may detect an event at any moment, and depending on the implemented application behaviour, it may want to notify the client immediately. At this moment – from the communications point of view – the server may be in one of the two following states:

a) no physical connection is established between the physical device containing the server, and any client device (but this connection to the client device can be established [22]);

b) there is already an established physical connection between the physical device containing the server and a client device.

In case a), in order to notify the event, the server shall first establish a physical connection. In case b), the following choices are available:

- the detected event may be reported via the existing connection;
- the current connection could be immediately aborted in order to start a new connection;
- the server may wait for the end of the current session before starting a new connection;
- etc.

Upon invocation of the EventNotification.request service, the COSEM server application layer shall build an EventNotification.request APDU. This APDU shall be sent from the SAP of the management logical device to the SAP of the client management application process, using data communication services of the lower layers, in a non-solicited manner.

In some protocol profiles, the lower protocol layers do not allow to provide a really unsolicited EventNotification service. In such cases, the client has to trigger the sending of the Event Notification.request APDU, using the Trigger_EventNotification_Sending.request service, defined in 6.5.4.2. The MSC shown in Figure 27 represents this case.

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

[22] 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.

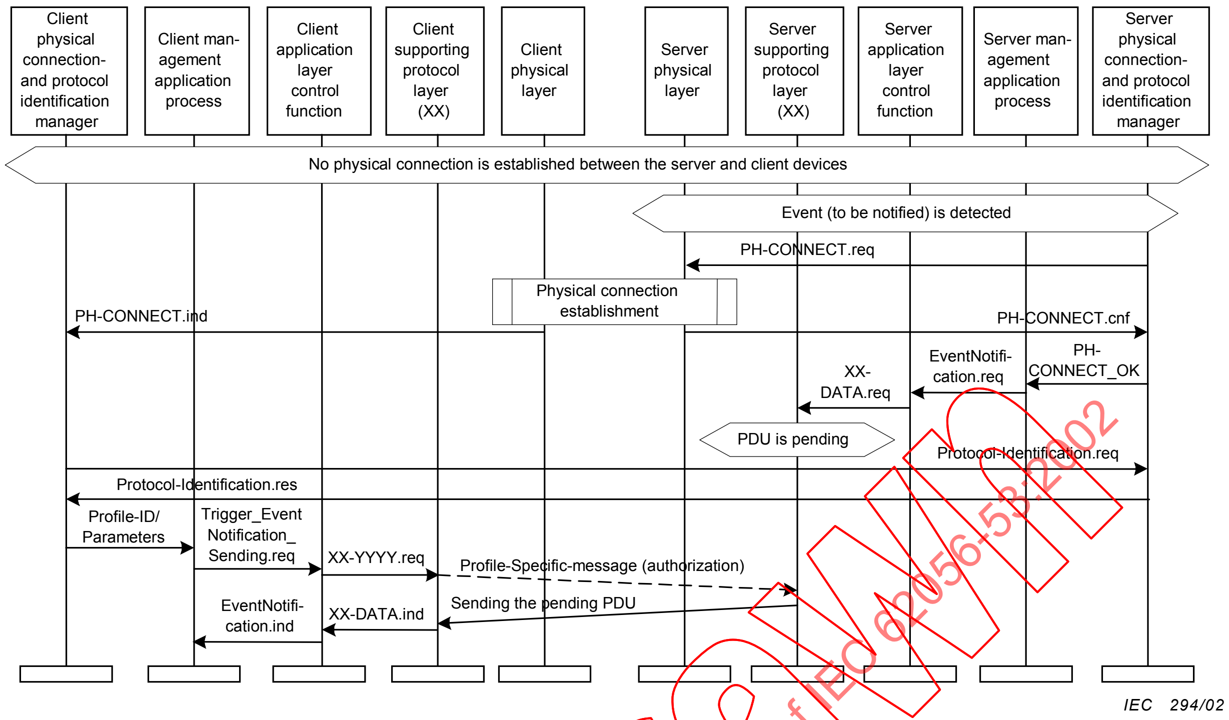

Client physical connection- and protocol identification manager
Client man-agement application process
Client application layer control function
Client supporting protocol layer (XX)
Client physical layer
Server physical layer
Server supporting protocol layer (XX)
Server application layer control function
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
XX-DATA.req
PDU is pending
Protocol-Identification.req
Protocol-Identification.res
Profile-ID/ Parameters
Trigger_Event Notification_ Sending.req
XX-YYYY.req
Profile-Specific-message (authorization)
Sending the pending PDU
XX-DATA.ind
EventNotifi-cation.ind
IEC 294/02

Figure 27 – 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 of the protocol stack.

Successful physical connection establishment is reported at both sides to the physical connection manager process. At the server side, this shall indicate to the COSEM application process, 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 XX-Data.request service of the supporting protocol layer with the data parameter carrying the Event-Notification-Request APDU.

At this moment, the supporting layer may not be able to send this PDU immediately, thus it will be stored in the server's supporting 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, which protocol profile is used by the calling server. Therefore, it has to identify the protocol stack using the protocol identification service described in IEC 62056-42. This is shown as a "Protocol-Identification.req" and a "Protocol-Identification.res" message in Figure 27. After the identification of the protocol profile used by the server, the client is able to instantiate the right protocol stack. If it is required in the given profile, the client shall invoke the Trigger_EventNotification_Sending.request service of the client application layer.

Upon invocation of this service, the client shall send its "authorization" to the server: this "authorization" message depends on the profile used. When this authorization is received, the server shall send the pending Event-Notification-Request APDU, using connectionless data transmission services. The received Event-Notification-Request APDU shall be indicated to the client application process as an EventNotification.indication. At this moment, the client is notified about the event, the sequence is completed.

The sequence described above is only an example: depending on the implemented behaviour, servers may report events in different circumstances. In any case, in order to notify the client about the detection of an event:

- the server shall use the EventNotification.request service invocation;
- this service invocation shall make the server application layer to build an EventNotification.request APDU;
- the resulting supporting layer PDU shall be sent at the first opportunity to the client, using the connectionless data communication services of the supporting layer. The nature of this first opportunity depends on the communications profile used;
- the received supporting layer PDU, which includes the EventNotification.request message, shall be indicated to the client application layer with the help of the connectionless XX-DATA.indication service. Upon reception of this service, the client application layer shall generate an EventNotification.indication[23] service to the COSEM client application;
- event notifications are always sent from the management logical device to the management application process.

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 service 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.

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[24]). However, it is possible to request to be served first by setting the priority service 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.

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

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

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.[25] These 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 service 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)[26] 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.

[25] 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.

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

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 if 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.

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 that which 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 specification – 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 application process shall assemble the requested data. If the data fit into one APDU, the server application process 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 application process 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 (page 93) with the following contents:

last-block (BOOLEAN)	= FALSE
block-number (Unsigned32)	= 0001
result (IMPLICIT OCTETSTRING)[27]	= 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 application process 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 application process 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 28 shows an example for the case, when the requested data can be sent in three parts, and the transfer is not aborted.

[27] It is the raw-data CHOICE. (see page 93).

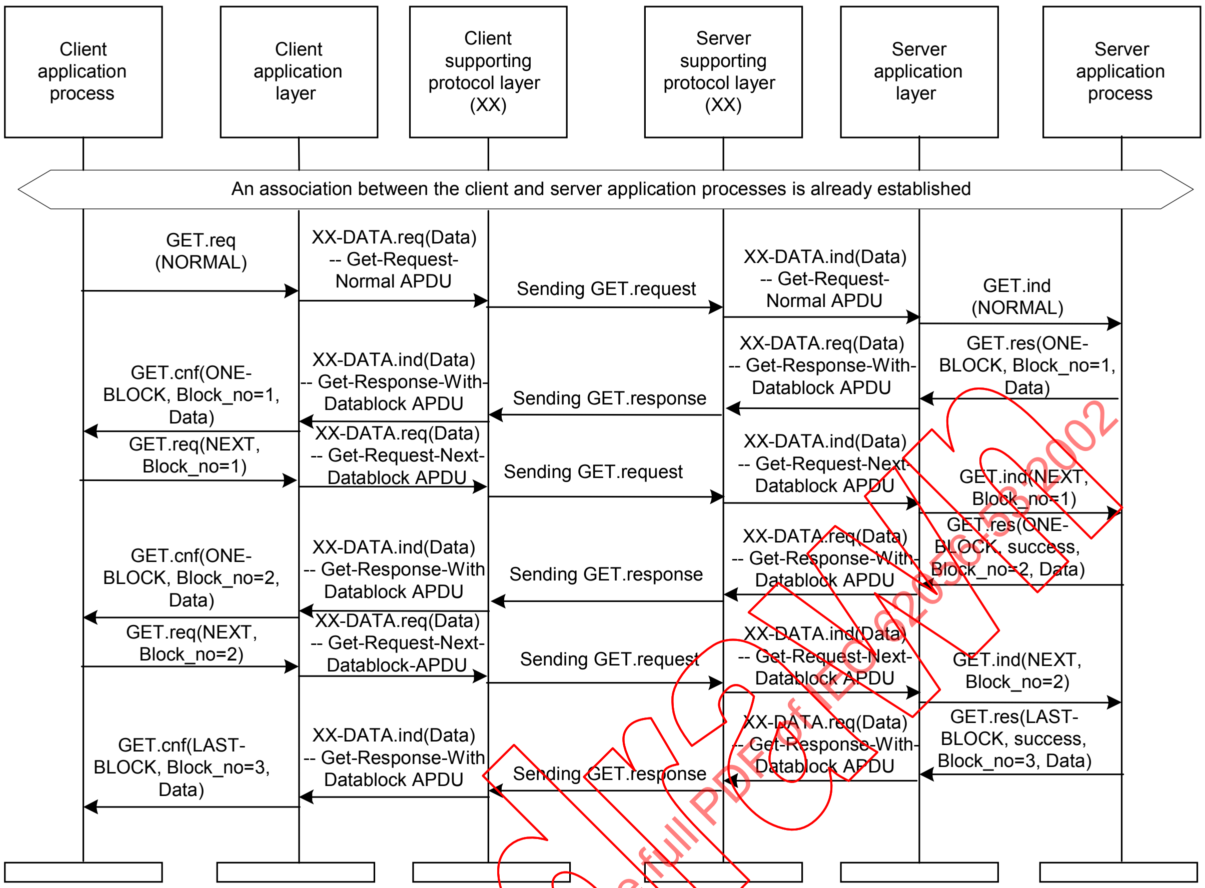

Client application process
Client application layer
Client supporting protocol layer (XX)
Server supporting protocol layer (XX)
Server application layer
Server application process
An association between the client and server application processes is already established
GET.req (NORMAL)
XX-DATA.req(Data) -- 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 295/02

Figure 28 – 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 expected by the client (received block-number + 1), and the result shall contain a data-access-result indicating the reason of 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-Next APDU 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-Next APDU 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.

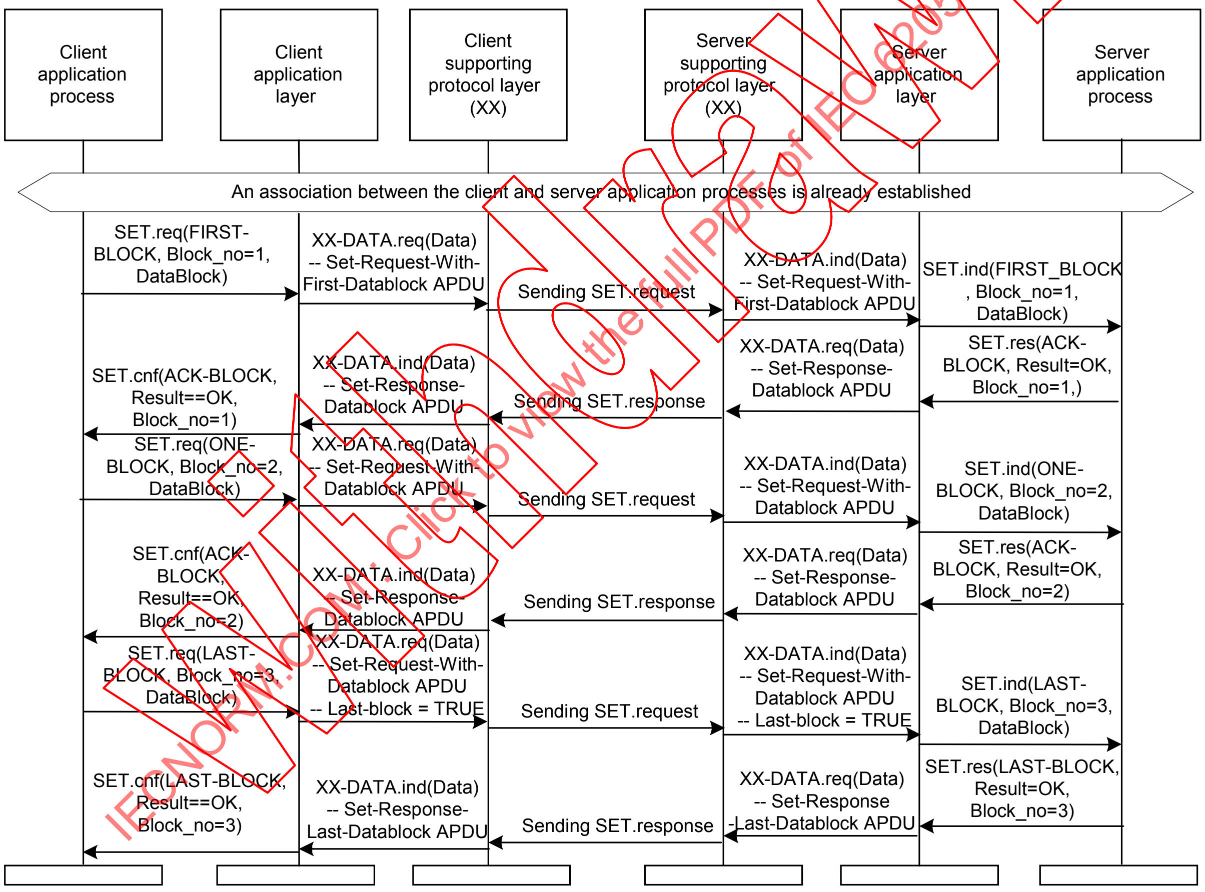
Block transfer with the GET service is a negotiable feature: its support is indicated by BIT 11 of the xDLMS conformance block.

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 296/02

Figure 29 – 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 chapter. 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.indication of type ONE BLOCK 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.

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 page 90.

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 30 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.

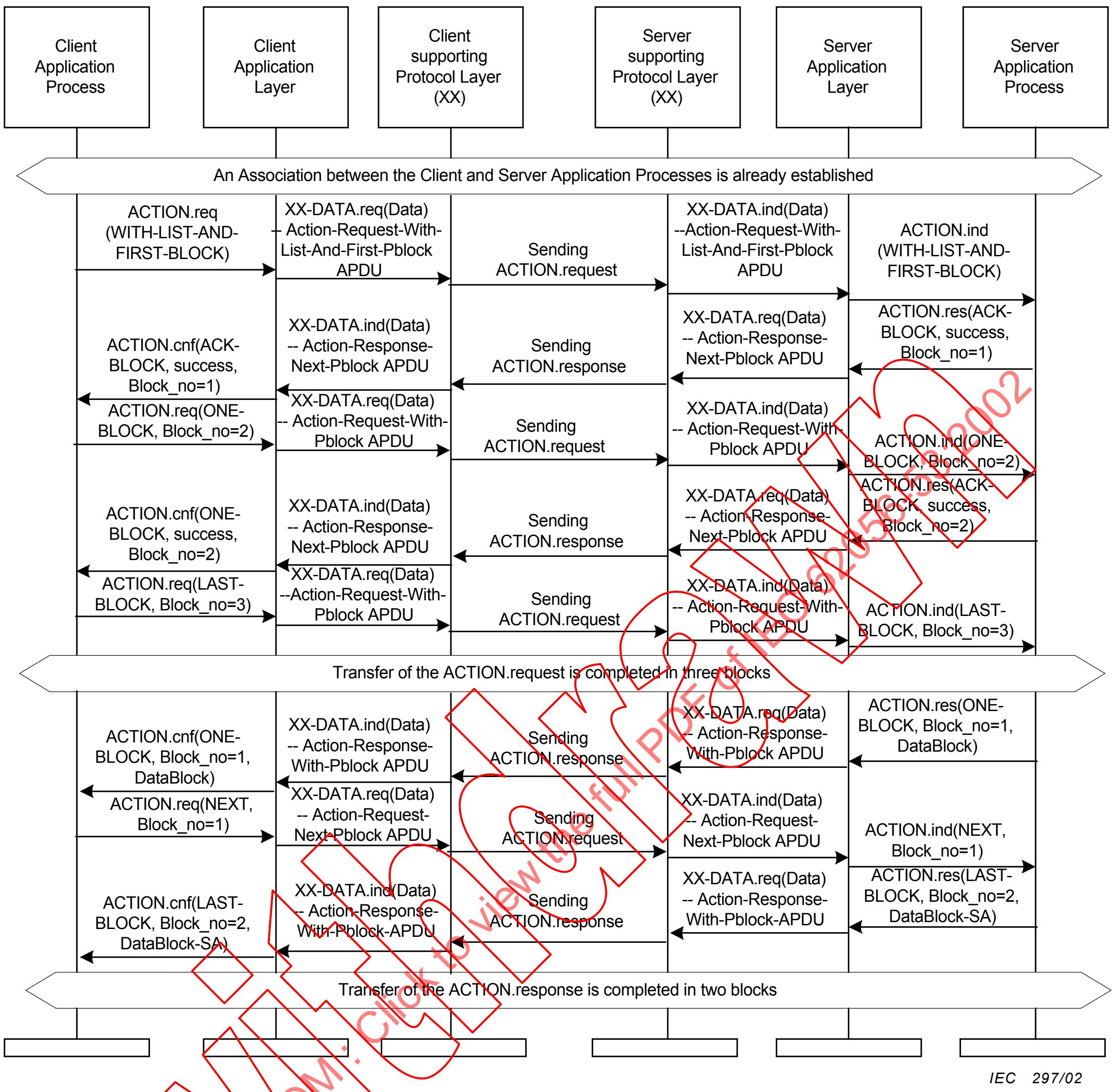

Client Application Process
Client Application Layer
Client supporting Protocol Layer (XX)
Server supporting Protocol Layer (XX)
Server Application Layer
Server Application Process
An Association between the Client and Server Application Processes is already established
ACTION.req (WITH-LIST-AND-FIRST-BLOCK)
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 297/02

Figure 30 – 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.

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 application process 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 application process 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 application process 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 31.

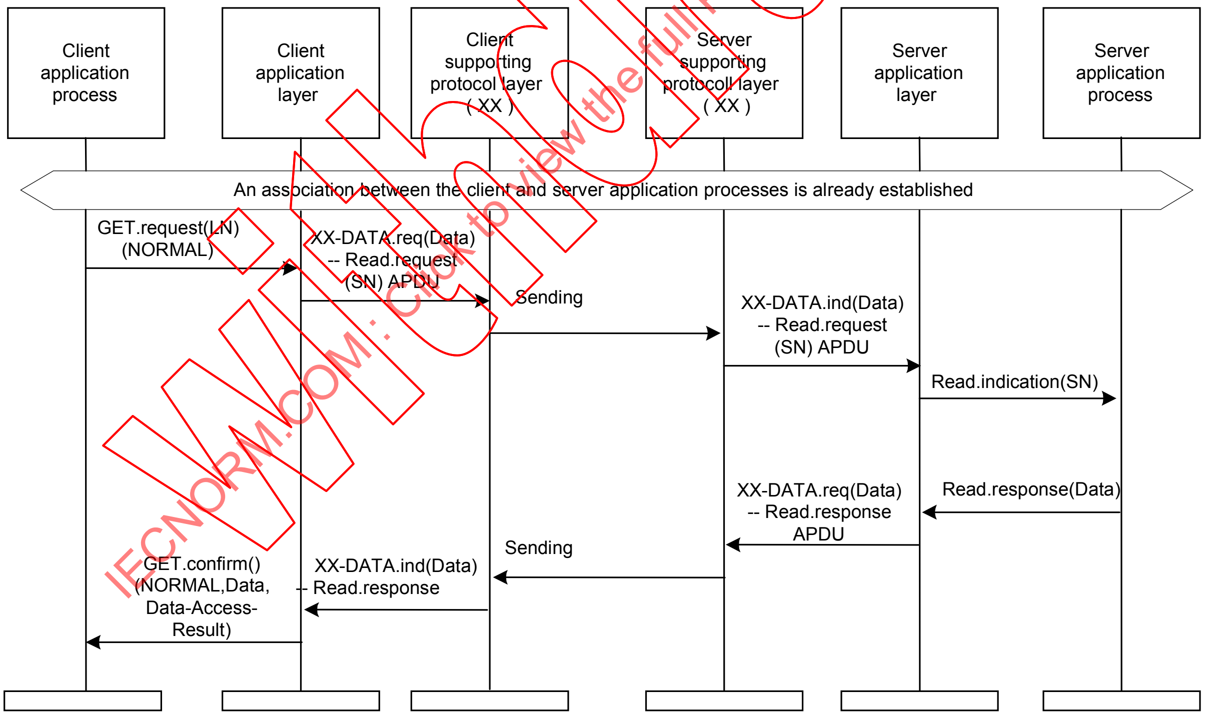

Client application process
Client application layer
Client supporting protocol layer (XX)
Server supporting protocoll layer (XX)
Server application layer
Server application process
An association between the client and server application processes is already established
GET.request(LN) (NORMAL)
XX-DATA.req(Data) -- Read.request (SN) APDU
Sending
XX-DATA.ind(Data) -- Read.request (SN) APDU
Read.indication(SN)
Read.response(Data)
XX-DATA.req(Data) -- Read.response APDU
Sending
XX-DATA.ind(Data) -- Read.response
GET.confirm() (NORMAL,Data, Data-Access-Result)
IEC 298/02

Figure 31 – 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 application association has already been established. Upon the receipt of this request, the client application layer shall invoke the connectionless data transfer service of the supporting protocol layer (XX-DATA.request) with the correctly formatted APDU as data parameter.

At the supporting layer level, the resulting XX-PDU shall be transmitted using the connection-less data transmission service to the indicated destination address. 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 application association 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.

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.

Events, like alarms, fraud detection, or simply a counter overflow generally occur asyn-chronously to any operation.

In order to ensure that the client is informed about such an event, a special, non- client/server type service, the InformationReport[28] service is available. Although any logical device of a server may have events to be reported, event notification in COSEM is the responsibility of the management logical device.

A server device may detect an event, which should be notified to a client, at any moment. At this moment, from the communications point of view, the server may be in two different states, as follows:

a) no physical connection is established between the physical device including the server, which detected the event and any client device (but this connection to the client device can be established [29]);
b) there is already an established physical connection between the physical device including the server, which detected the event, and a client application.

[28] When logical name referencing is used, the service is called EventNotification at the server side. At the client's side the received InformationReport SN service is mapped to an EventNotification.indication service as described below.

[29] Physical connection cannot be established when the server has only a local interface (e.g. IEC 62056-21 optical link), and the hand held terminal, housing the client application is not connected, or the telephone line is not available for a server which has a PSTN connection. Handling such cases is implementation specific.

In case a), in order to notify the event, the server shall first establish a physical connection. In case b) the following choices are available:

- the detected event might be reported via the existing connection;
- the current connection could be immediately aborted in order to start a new connection;
- the server may wait for the end of the current session before starting a new connection;
- etc.

Upon invocation of the InformationReport.request service, the COSEM server application shall build an InformationReport.request APDU. This APDU shall be sent from the SAP of the management logical device to the SAP of the client management device, using of data services of the lower layers, in a non-solicited manner.

In some profiles the lower layers do not allow to provide a really non-solicited InformationReport.request service. In such cases, the client shall trigger the sending of the InformationReport.request APDU in using the Trigger_EventNotification_Sending.request service.

Figure 27 shows an example for the EventNotification service: this example also applies to the InformationReport service. The difference is that in the case of SN referencing, the server, instead invoking the EventNotification.request service, shall invoke the Information Report.request service of the server side application layer, and, of course, the transmitted APDU shall correspond to this, InformationReport service. At the client side there is no change: upon the receipt of an InformationReport APDU, the client application layer shall generate an EventNotification.indication primitive to the client application process.

7.4.2.4 Mapping of an InformationReport service to a 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:

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 follows:

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

The meaning of the access_selector and of the parameter depends on the variable referenced. It is defined in IEC 62056-62.

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

Long service parameters are transmitted using the segmentation feature provided by the data link layer as described in IEC 62056-46. (Only transparent long data transfer is allowed with SN referencing.)

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 selected30)
      initiateRequest                [1] IMPLICIT        InitiateRequest,
      readRequest                    [5] IMPLICIT        ReadRequest,
      writeRequest                   [6] IMPLICIT        WriteRequest,

      initiateResponse               [8] IMPLICIT        InitiateResponse,
      readResponse                   [12] IMPLICIT       ReadResponse,
      writeResponse                  [13] IMPLICIT       WriteResponse,

      confirmedServiceError          [14]                ConfirmedServiceError,

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

-- the two ACSE APDUs
      aarq       AARQ-apdu
      aare       AARE-apdu,

-- 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
```

30 Ciphered application contexts will use the corresponding ciphered DLMS PDUs.

8.2 The AARQ and AARE APDUs

```
AARQ-apdu    ::= [APPLICATION 0] IMPLICIT SEQUENCE
                                        -- [APPLICATION 0] == [ 60H ] = [ 96 ]
 {
    protocol-version                [0] IMPLICIT BIT STRING {version1 (0)} DEFAULT {version1},
    application-context-name        [1] Application-context-name,
    called-AP-title                 [2] AP-title OPTIONAL,
    called-AE-qualifier             [3] AE-qualifier OPTIONAL,
    called-AP-invocation-id         [4] AP-invocation-identifier OPTIONAL,
    called-AE-invocation-id         [5] AE-invocation-identifier OPTIONAL,
    calling-AP-title                [6] AP-title OPTIONAL,
    calling-AE-qualifier            [7] AE-qualifier OPTIONAL,
    calling-AP-invocation-id        [8] AP-invocation-identifier OPTIONAL,
    calling-AE-invocation-id        [9] AE-invocation-identifier OPTIONAL,
    -- The following field shall not be present if only the kernel is used.
    sender-acse-requirements        [10] IMPLICIT ACSE-requirements OPTIONAL,
    -- The following field shall only be present if the authentication functional unit is selected.
    mechanism-name                  [11] IMPLICIT Mechanism-name OPTIONAL,
    -- The following field shall only be present if the authentication functional unit is selected.
    calling-authentication-value    [12] EXPLICIT Authentication-value OPTIONAL,
    implementation-information      [29] IMPLICIT Implementation-data OPTIONAL,
    user-information                [30] 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
 }

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[31]

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)
          }
  }
```

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…6),
                                        priority        (7)
                                 }

ObjectName    ::= Integer16

Cosem-Class-Id                  ::= Unsigned16

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

[31] In ISO/IEC/TR2 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.

```
Cosem-Object-Attribute-Id      ::= Integer8

Cosem-Object-Method-Id         ::= Integer8

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
  }

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,[32]
         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,
```

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

```
        makeUsable             [13]      ServiceError,
        initiateLoad           [14]      ServiceError,
        loadSegment            [15]      ServiceError,
        terminateLoad          [16]      ServiceError
        initiateUpLoad         [17]      ServiceError,
        upLoadSegment          [18]      ServiceError,
        terminateUpLoad        [19]      ServiceError
  }

ServiceError    ::= CHOICE
  {
        application-reference  [0]               IMPLICIT ENUMERATED {
           -- DLMS provider only
              other                               (0),
              time-elapsed                        (1),     -- time out since request sent
              application-unreachable             (2),     -- peer AEi not reachable
              application-reference-invalid       (3),     -- addressing trouble
              application-context-unsupported     (4),     -- application-context incompatibility
              provider-communication-error        (5),     -- error at the local or distant equipment
              deciphering-error                   (6)      -- error detected by the deciphering function
           },
        hardware-resource      [1]                        IMPLICIT ENUMERATED {
             -- VDE hardware troubles
              other                                        (0),
              memory-unavailable                           (1),
              processor-resource-unavailable               (2),
              mass-storage-unavailable                     (3),
              other-resource-unavailable                   (4)
           },
        vde-state-error        [2]                        IMPLICIT ENUMERATED {
             -- Error source description
              other                                        (0),
              no-dlms-context                              (1),
              loading-data-set                             (2),
              status-nochange                              (3),
              status-inoperable                            (4)
           },
        service                [3]               IMPLICIT ENUMERATED {
             -- service handling troubles
              other                               (0),
              pdu-size                            (1),   --  pdu too long
                                                         --  (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,
        floating-point          [7]    IMPLICIT OCTET STRING(SIZE(4))33,
        octet-string            [9]    IMPLICIT OCTET STRING,
        visible-string          [10]   IMPLICIT VisibleString,
        time                    [11]   IMPLICIT GeneralizedTime,
        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)),
```

[33] The four bytes of this OCTET STRING shall contain a floating-point value formatted as it is specified as Short Floating-Point Number format in IEEE Standard 754-1985.

```
        float64                 [24]    IMPLICIT OCTET STRING (SIZE(8)),
        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,
        floating-point          [7]     IMPLICIT NULL,
        octet-string            [9]     IMPLICIT NULL,
        visible-string          [10]    IMPLICIT NULL,
        time                    [11]    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,
        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 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-received-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
  }
```

NOTE In COSEM, the quality-of-service parameter is not used. The meter 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 B.

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[34]
  }
```

[34] There shall 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** shall be present as null_data.

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

Action-Request-With-List-And-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
  {
        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                                  OCTET STRING[35] OPTIONAL,
        cosem-attribute-descriptor            Cosem-Attribute-Descriptor,
        attribute-value                       Data
  }
```

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

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 3-layer, connection-oriented, HDLC based profile

A.1 Introduction

The COSEM application layer is the only layer which contains COSEM specific service element, the extended DLMS application service element (xDLMS-ASE).

This application layer may be used with a variety of lower layer protocols to accomplish the communication function. A full protocol stack – including the application layer – is called communication profile (see clause 4).

A communications profile is characterized by:

- the lower protocol layers;
- the type (connection-oriented or connectionless) of the application control service element (ACSE) included in the application layer.

The first communication profile specified and standardized for COSEM is the 3-layer, connection-oriented, HDLC based profile. This profile consists of three protocol layers:

- the COSEM application layer, including the connection oriented ACSE in the application layer, as defined in this standard;
- the data link layer, based on the ISO/IEC 13239 HDLC protocol, as defined in IEC 62056-46,
- the physical interface layer, as defined in IEC 62056-42.

A.2 The HDLC-based data link layer – Overview

For the purposes of this profile, the following selections from the HDLC standard ISO/IEC 13239 have been made:

- unbalanced connection-mode data link operation[36];
- two-way alternate data transfer;
- the selected HDLC class of procedure is UNC (Unbalanced operation Normal response mode Class), extended with UI frames;
- frame format type 3;
- non-basic frame format transparency.

In the unbalanced connection-mode data link operation, two or more stations are involved. The primary station assumes responsibility for the organization of data flow and for unrecoverable data link level error conditions by sending command frames. The secondary station(s) respond by sending response frames.

The basic repertoire of commands and responses of the UNC class of procedures is extended with the Unnumbered Information (UI) frame to support connectionless data communication services, in order to provide multicasting and broadcasting and non-solicited information transfer from server to the client.

[36] In COSEM, the primary station corresponds to the client application, and the secondary station corresponds to the secondary station

Using the unbalanced connection-mode data link operation implies that the client and server side data link layers are different in terms of the sets of HDLC frames and their state machines.

A.2.1 Services of the HDLC based data link layer

This HDLC based data link layer provides services for:

- data link layer connection management;
- connection oriented data communication (I frames);
- connectionless data communication (UI frames).

A.2.1.1 Services at the client side

Figure A.1 summarizes the data link layer services used by the COSEM client application layer, in the case of the three-layer, HDLC-based, CO communication profile.

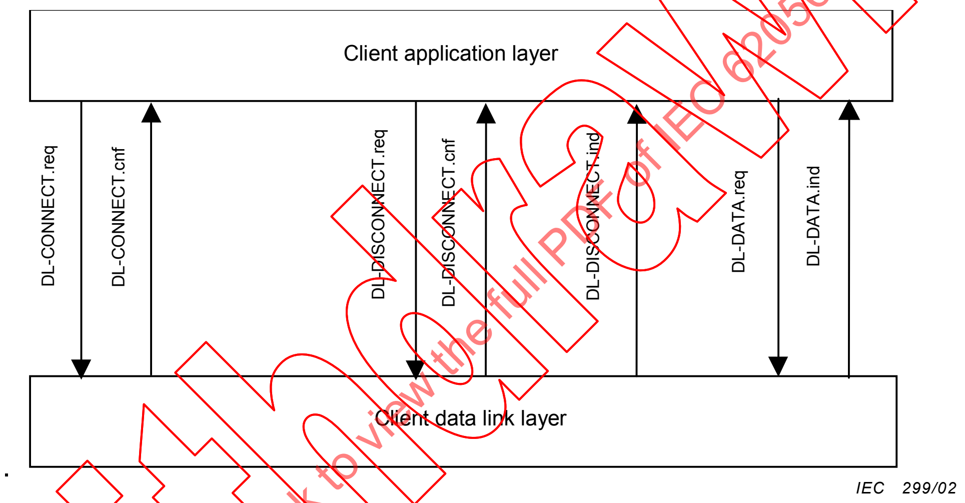


Figure A.1 – Data link services used by the client COSEM application layer

For some services, the correspondence between an application layer (ASO) service invocation and the supporting data link layer service invocation is straightforward (e.g. invoking the COSEM-OPEN.request service directly implies the invocation of a DL-CONNECT.request service). For other services, this direct service mapping cannot be established.

A.2.1.2 Services at the server side

Figure A.2 summarizes the data link layer services used by the COSEM server application layer, in the case of the three-layer, HDLC based, CO communications profile.